A Direct Approach for the Segmentation of Unorganized Points and Recognition of Simple Algebraic Surfaces

PhD Thesis

M. Vančo

University of Technology Chemnitz

October 2002

Ein direktes Verfahren zur Segmentierung unstrukturierter Punktdaten und Bestimmung algebraischer Oberflächenelemente

Dissertation
zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

vorgelegt
der Fakultät für Informatik
der Technischen Universität Chemnitz

von:	Dipl.-Inf.(SK) Marek Vančo
geboren am:	6. Juni 1974
in:	Bratislava

Gutachter:	Prof. Dr. Guido Brunnett
	Prof. Dr. Beat D. Brüderlin
	Prof. Dr. Hans Hagen

Chemnitz, 10. Oktober 2002

Abstract

In *Reverse Engineering* a physical object is digitally reconstructed from a set of boundary points. In the segmentation phase these points are grouped into subsets to facilitate consecutive steps as surface fitting. In this thesis we present a segmentation method with subsequent classification of simple algebraic surfaces. Our method is direct in the sense that it operates directly on the point set in contrast to other approaches that are based on a triangulation of the data set.

The reconstruction process involves a fast algorithm for k-nearest neighbors search and an estimation of first and second order surface properties. The first order segmentation, that is based on normal vectors, provides an initial subdivision of the surface and detects sharp edges as well as flat or highly curved areas. One of the main features of our method is to proceed by alternating the steps of segmentation and normal vector estimation. The second order segmentation subdivides the surface according to principal curvatures and provides a sufficient foundation for the classification of simple algebraic surfaces. If the boundary of the original object contains such surfaces the segmentation is optimized based on the result of a surface fitting procedure.

Zusammenfassung

Im *Reverse Engineering* wird ein existierendes Objekt aus einer Menge von Oberflächenpunkten digital rekonstruiert. Während der Segmentierungsphase werden diese Punkte in Teilmengen zusammengefügt, um die nachfolgenden Schritte wie *Flächenerkennung* (surface fitting) zu vereinfachen. Wir präsentieren in dieser Arbeit eine Methode zur Segmentierung der Punkte und die anschließende Klassifikation einfacher algebraischen Flächen. Unser Verfahren ist direkt in dem Sinne, dass es direkt an den Punkten arbeitet, im Gegensatz zu anderen Verfahren, die auf einer Triangulierung der Punktmenge basieren.

Der Rekonstruktionsprozess schließt einen neuen Algorithmus zur Berechnung der k-nächsten Nachbarn eines Oberflächenpunktes und Verfahren zur Schätzung der Flächeneigenschaften ersten und zweiten Grades ein. Die normalenbasierte Segmentierung (Segmentierung ersten Grades) liefert eine Aufteilung des Objektes und detekiert scharfe Kanten, sowie flache oder stark gekrümmte Gebiete des Objektes. Ein zentrales Element unserer Methode ist die Wiederholung der Schritte der Segmentierung und der Schätzung der Normalen. Erst die Iteration ermöglicht die Schätzung der Normalen in der benötigten Genauigkeit und die Generierung einer zufriedenstellender Segmentierung. Die Segmentierung zweiten Grades teilt die Oberfläche nach den Hauptkrümmungen auf und bietet eine zuverlässige Grundlage für die Klassifizierung einfacher algebraischen Flächen. Falls der Rand des Ausgangsobjektes solche Flächen enthält, wird der Segmentierungsprozess auf der Grundlage des Ergebnisses der Flächenerkennungsprozedur optimiert.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The computers affected the human life in almost all areas. Especially in the industry they became an essential part of the manufacturing process - they are used e.g. for path planning in numerical controlled machining, quality controlling, CAD/CAM (Computer Aided Design / Computer Aided Manufacturing), visualization etc. The need for digital models used in the manufacturing process caused a rapid development of *reverse engineering*.

There are numerous real objects, for which the computer representation is demanded. The reason is obvious: the computer representation is compact, easy to transfer without loss of information (thanks to Internet). Together with an appropriate tool the digitized model can easily be visualized, is simple to modify (e.g. CAD representation can be very easy modified) etc. To get a computer representation of an object there two quite different ways:

- To model the object in a modeler program (e.g. AutoCAD, Pro Engineer, Catia, Surfacer etc. for creating a CAD model or 3D Studio Max, SoftImage, Maya, TrueSpace etc. for general modeling).
- To scan the object with a 3D scanner and to reconstruct (if possible) the surface with a *surface reconstruction* program.

The first choice (to model the object in a modeler program) is the direct way (so-called *forward method*) to obtain a computer representation. The greatest disadvantage is, that for a complex object this method can be extremely time consuming and require that the human is very familiar with the modeling program.

The disadvantage of the second choice (the *reverse method*) is that a 3D scanner is not (at present) a very common computer equipment and the high-end accurate 3D scanners are rather expensive. The advantage is a slight user interaction (the aim of surface reconstruction tools is a full automatic program with no user interaction). For complex objects the reconstruction tools are mathematically very difficult and the authors believe that some user assistance will always be necessary.

In this thesis we present an approach for automatic surface segmentation and recognition of sampled objects, consisting of unorganized boundary points. Our aim is the

development of an automatic (or a semi-automatic) tool, which is able to decompose the object into coherent point clusters based on estimated surface properties. These clusters are hereafter approximated by surfaces and the surface parameters are obtained. Using an existing tool (e.g. ACIS library) boundary curves of the recognized surfaces or intersection curves between two surfaces can be approximated, which together with the provided surface decomposition fully describe the object to be reconstructed.

In the beginning the user should provide starting parameters for the whole reconstruction process. No further user interaction is desired in the future work, but at the moment the user should supervise the reconstruction process and, if necessary, slightly modify the initial parameters and repeat the failed reconstruction step.

We based the whole segmentation and reconstruction on a general data structure, which can efficiently be computed even for large data sets: the *neighborhood graph*. The description of our new method for fast finding of *k*-nearest neighbors to a given point is shown in the chapter 3. The method is based on a median subdivision of the space containing the points (creating so-called *buckets*) and the searching in each bucket is accelerated by a hashing strategy with an appropriate distribution function. Further we introduce three new methods for computation of the surface normal vectors from the neighborhood graph and discuss their stability and efficiency. A very fast modification of the well known orientation propagation algorithm for consistent normal vectors orientation is presented. At the end of the chapter 3 the method for second order surface properties estimation is described.

In the chapter 4 we present our *first* and *second order surface segmentation*. In the first step that is based on normal vectors propagation, sharp or nearly sharp edges (i.e. edges with a very high variation of normal vectors) in the object are detected. The detection allows us to improve our initial normal vectors estimation by modifying the neighborhood of the points close to the sharp edges. In the second order segmentation principal curvatures are used to subdivide the data set. Here, tangent continuous but curvature discontinuous edges are detected. The segmentation procedure is able to find and separate parts of simple algebraic surfaces (planes, cylinders, cone, spheres). To keep the user interaction and the number of repetitions as small as possible we provide to the user an estimation of the segmentation parameters, which are crucial for a proper second order segmentation. The method how to estimate these parameters from the first order segmentation is given at the end of the chapter 4.

The task of the second order segmentation is to subdivide the object according to the specified criteria into “consistent” parts. These parts do not have to correspond to the fundamental geometric surfaces (as planes, spheres, etc.), which form the object. It even happens, that if the object contains some sampling flaws or noise, one simple algebraic surface can be subdivided into many segments. An important part of this thesis is the *postprocessing* (chapter 5) of the segmentation, which provides an improvement of the segmentation based on the surface recognition. Its task is to find the segments that lie on the same simple algebraic surface (as planes, cones, cylinders or spheres), fit them to the simple surface and extend the recognized segments. The postprocessing consists of the steps of

- **segment classification**: an assumption about the geometric surface type of a segment is made and validated by creating an appropriate parameterization and by fitting an algebraic surface to the segment. Finally the surface parameters are obtained.

- **segment extension**: neighboring points are added to a segment, if they fulfill the surface equation within a prescribed tolerance.

Similarly, as in the previous chapter, at the end of the chapter 5 we introduce our procedure for automatic postprocessing thresholds estimation, in order to reduce the necessary user assistance.

Our reconstruction method is very fast (the whole segmentation is done within a few seconds for a data set of 100 thousand points), which allows the user, if necessary, to fine tune the reconstruction results. In the chapter 6 results obtained by our method are presented for various objects.

Chapter 2

Background and Fundamentals of *Reverse Engineering*

Reverse engineering and *surface reconstruction* are young fields of the computer graphics. They got very popular in early 80's with rapid development of the object scanning devices.

There is a vast amount of objects (objects in real life, parts of machines, toys etc.), for which the computer representation is not known (like hand modeled prototypes) or not available. The aim of the surface reconstruction is to transform such an object into digital representation, which can be used for further visualization, processing, manufacturing etc.

The methods of 3D reconstruction and the further processing of the point clouds have already been introduced in almost all fields of human life such as medicine, architecture, machine building, die making industry, design, toy industry as well as clothing. The entertainment industry (film making industry), scientific visualization as well as virtual reality worlds can also be mentioned as fields demanding 3D digitized objects.

The analogy between 2D and 3D object reconstruction is illustrated in the fig. 2.1: for 2D "reconstruction" the raw data are paper documents (text or photos), which are digitized with an usual optical scanner, processed (e.g. OCR: transformation of an image into a text), optionally modified and prepared for printing or plotting. Analogously, a 3D object is scanned by a scanning device (for details see section 2.1), reconstructed (a computer representation is created, which fully covers the initial object and is easy to handle, modify, test, process etc.) and can optionally be manufactured.

The whole process can be subdivided into following steps:

- **Data acquisition** - object scanning.
- **Multiview registration** - aligning the scanned views.
- **Surface reconstruction** - creating a computer model (CAD, polygonal approximation etc.).
- **Manufacturing** - [optional]

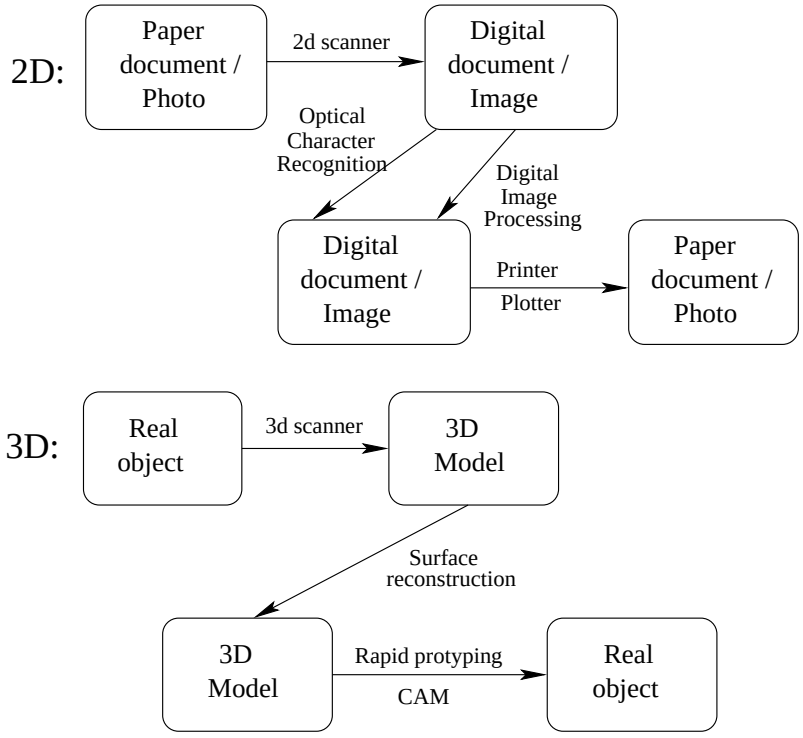


Figure 2.1: 2D and 3D object reconstruction

2.1 Data acquisition

The acquisition of the geometry of 3D models is a longstanding difficult problem in computer graphics. In the past, forward methods such as CSG, B-rep, NURBS, and sweeping have been successful in modeling regular shapes such as mechanical and architectural parts. These methods allow a user to directly and interactively define the shape of an object. On the other hand, natural objects and scenes often have irregular shapes and complex surface structures. It is very difficult to use forward methods to interactively define the details of the complex shapes and surfaces.

Further, the desire to reduce the dependence on human input in making realistic images of complex scenes has, over the past ten years, resulted in an increased role for measurements of the real world in the computer graphics pipeline. With the availability of vision equipment and the maturity of computer vision technology, there is now a trend to reverse engineer the acquisition process by recovering the geometric and topological information from measurements of real scenes and objects.

The scanning process can be divided into two categories:

- Passive scanning (sensing).

- Active scanning (sensing).

A few years ago computer vision scientists were mainly interested in **passive sensing** techniques that were supposed to reflect the way the human eye is working, i.e. no energy is emitted for the purpose of sensing, it is only received. Such passive techniques include stereo vision and monocular techniques like *shape-from-contour* or *shape-from-shading*. The problem is that recovering 3D information from a single 2D image is an ill posed problem. Thus, monocular techniques have to add a priori information such as surface smoothness to recover 3D data - a process known as regularization. Regularization is most often context dependent and is difficult to apply outside of the lab. Stereo vision techniques use two or more sensors, and thus two or more 2D images to recover 3D information. To interpret disparity between images, the matching problem must be solved, which has been formulated as an ill-posed problem too in a general context and which anyway is a task difficult to automate.

To overcome the above problems of passive sensing, **active sensing** techniques have been developed, i.e. properly formatted light (or any other form of energy) is emitted and then received once it has interacted with the object to digitize. Typically, light is emitted in the direction of an object, reflected on its surface and recovered by the sensor and the distance to the surface is calculated. Obviously, volume digitization techniques like computerized tomography (CT), magnetic resonance imaging (MRI) or ultrasound imaging also fall in this category.

Scanners, based on the type of acquiring the information from a 3D object can be categorized into these groups:

- Contact scanners:
  - Touch probe scanners
- Non contact scanners:
  - Laser scanners
  - Optical scanners using structured light

A *touch probe scanner* (sometimes called *coordinate measuring machines* - CMM), see fig 2.2 physically touches the object to be scanned with a probe. The probe is usually mounted on a bench which must have enough degrees of freedom to move around the part to digitize. The probe is either moved manually or the bench has motors and digital command control. Most CMMs translate along three orthogonal axes. Manual articulated arms have recently appeared at the market and offer an easier to use, although less precise, alternative.

CMMs are very precise and efficient, but their biggest disadvantage is the speed - they work very slow. They digitize one sample at a time, and for each sample the probe has to carefully approach the surface, contact it, then reach for the next sample. Although path planning software tools for CMMs do exist - which make the task considerably easier - there were no tools to visualize and process the large sets of samples - ‘clouds of points’ - involved in the digitization of complex surfaces.

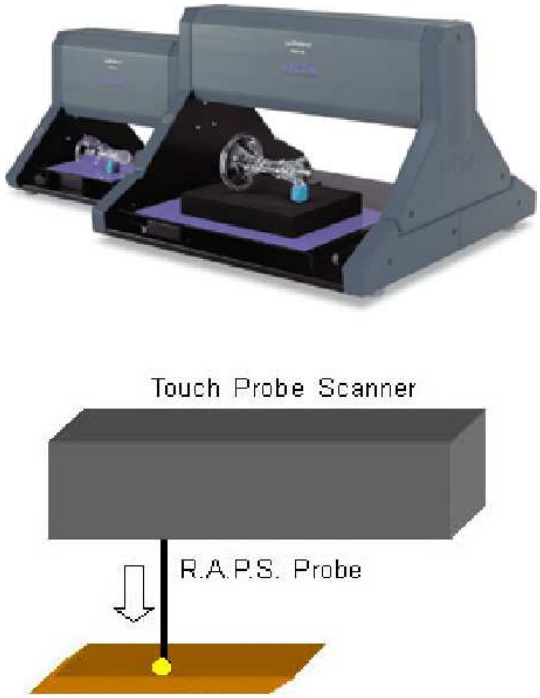


Figure 2.2: Touch probe scanner

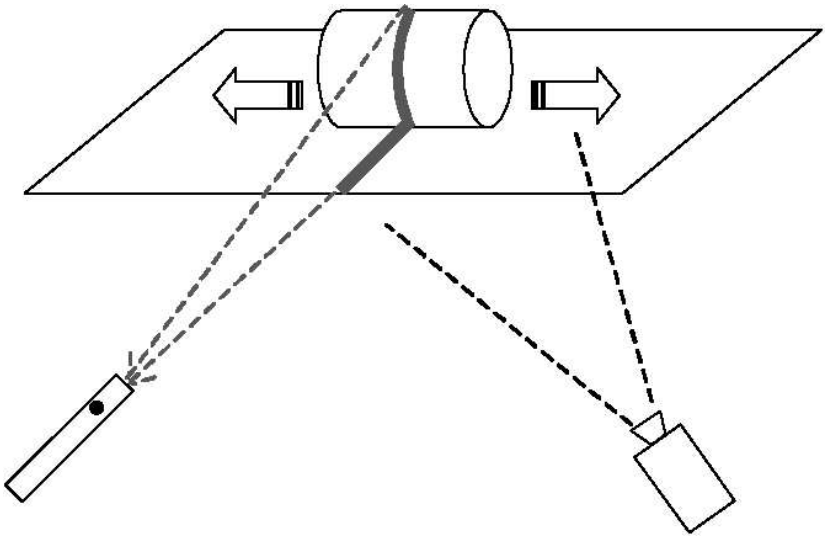


Figure 2.3: Laser scanner

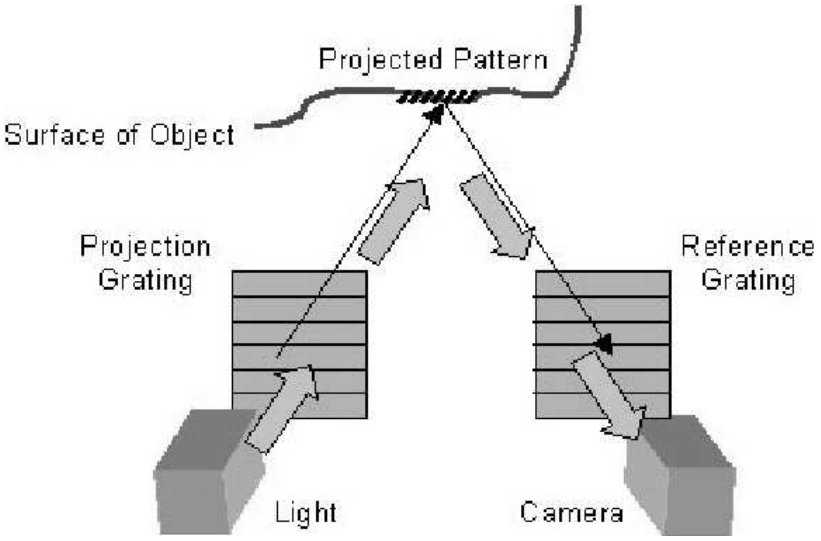


Figure 2.4: Optical Scanner

The expansion of computer-aided mechanical design and computed-aided manufacturing (CAD/CAM) has brought new needs. Digitizing a few samples may be enough to measure simple geometric surfaces like perfect cylinders, but modeling complex surfaces means digitizing thousands, sometimes millions of samples.

Among the active non-contact devices for 3D measurements, the *laser scanner* is probably the most versatile one. Its main advantages are great setup simplicity, small size and high resolution.

There are a variety of ways of using lasers to measure distance. The precise 3D shape or profile of solid objects can be determined using laser scanning techniques. Common approaches include:

- **Time of flight:** it measures the time the light travels to the scanned object. It is probably the most difficult, especially for short to medium distances where high resolution is desired. The simplistic method of just measuring the round trip from the reflection of a pulsed laser beam requires extraordinarily precise timing as the range decreases and the desired resolution increases.

- **Chirped pulse lidar/radar:** the laser transmitter sends out an optical signal which include a subcarrier with a time-varying (chirped) frequency. If the frequency changes linearly, the difference between the outgoing and detected signal frequencies $\Delta\mathbf{f}$ is a constant (over the duration of the chirp) and the distance is then:

$$\text{Distance} = \frac{\Delta\mathbf{f}\, c}{2\text{chirp\_rate}}$$

  Where c is the velocity of light.

- **Triangulation:** a light source illuminates an object and an image of this light spot is then formed, by means of a lens, on the surface of a linear light sensitive sensor. By measuring the location of the light spot image the distance of the object from the instrument can be determined. Common optical range finders can measure the distance using the difference in angles of the line-of-sight from an LED to the scene and back to a near photodetector. As the distance is reduced, the angle increases. This is coupled to the lens focusing mechanism. The use of a well collimated laser would increase both the maximum useful distance and resolution of such a system.

- **Interferometry:** for very precise measurement of small changes in position, the use of the wave nature of coherent laser light cannot be surpassed. Resolution scaled down to a few nanometers is possible with relatively simple equipment.

Nowadays, there are two main approaches to the rapid acquisition of the models of natural objects and scenes: *structured light* and *multiple views*. The first approach projects a structured light pattern, usually a stripe of laser beam, onto the surface of an object and uses a camera to capture the surface contour as revealed by the reflected laser light. The contours along different surface patches are then combined to compute the 3D geometry of the object. Such laser scanning products are currently available in two versions: hand-held and automated. The hand-held version requires the user to manually move the laser stripe to various parts of the object's surface. It, therefore, takes a lot of manual work to capture all the surface details of an object. The automated version uses a motorized system to move the laser in a manner similar to a 2D image scanner. 3D laser scanners typically produce accurate results but are expensive, especially the automated version and the color scanners.

The second approach uses one or more cameras to capture multiple views of an object and uses the multiple views to recover 3D coordinates of the feature points on the object's surface. The single-camera system either moves the camera or the object so as to capture various views of the object. The multiple-camera system typically employs static cameras to simultaneously capture multiple views of the object. However, it would require many cameras to adequately capture various facets of a complex object, and would therefore be more complex and expensive than the single-camera system.

Example of a laser scanning device:

Laser beam and laser plane triangulation sensors Cyberware 3D color digitizers are very often introduced in the movie industry. The sensor is mounted on proprietary benches and translates/rotates along digitized surfaces (e.g. actors) to perform rectangular/cylindrical scans. The sensor projects a fine line of laser light across the surface. The shape of this line is captured and processed to recover the depth of every sample. As the sensor moves along, the same line is illuminated with white light and a color video camera captures the apparent 'color' - texture.

2.2 Multiview registration

The goal of registration is to transform sets of surface measurements into a common coordinate system. To capture a complete object surfaces multiple 2.5D range images from different viewpoints are required as illustrated in the fig. 2.5.

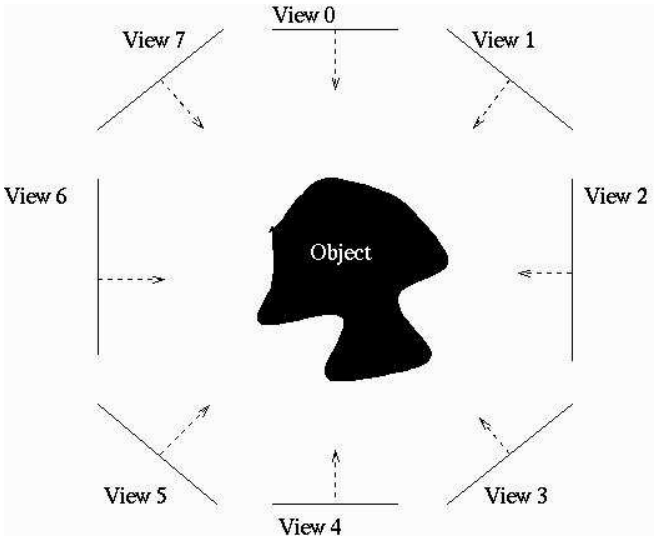


Figure 2.5: Multiview scanning

Besl and McKay [10] introduced the *Iterative Closest Point* (ICP) algorithm to register two sets of points on a free-form surface. ICP is a general purpose, representation independent method for the accurate and computationally efficient registration of 3D shapes including free-form curves and surfaces. Extensions of this algorithm are now widely used for registration of multiple sets of surface data. The original algorithm registers two point sets provided one is a subset of the other and that the transform between the sets is approximately known. The original ICP algorithm operates as follows given point set $\mathcal{P}$ and surface Q where $\mathcal{P}$ is a subset of Q:

- **Nearest point search**: For each point p in $\mathcal{P}$ find the closest point q on Q.
- **Compute registration**: Evaluate the rigid transform T that minimizes the sum of squared distances between pairs of closest points (p,q).
- **Transform**: Apply the rigid transform T to all points in set $\mathcal{P}$.
- **Iterate**: Repeat step 1 to 3 until convergence.

This approach will converge to the nearest local minimum of the sum of squared distances between closest points. A good initial estimate of the transformation between point sets is required to ensure convergence to the correct registration. Incorrect registration may occur if the error in the initial transformation is too large (typically

greater than 20°) or if the surface does not contain sufficient shape information for convergence.

Modifications to the original ICP algorithm have been made to improve the rate of convergence and register partially overlapping sets of points, e.g. Chen and Medioni [15] demonstrated the registration of partially overlapping range image data. A modified cost function was used to compute the registration which minimizes the squared distance in the direction of the surface normal. This cost function gives improved rates of convergence.

Multiple range images are often required to capture the complete surface of an object. The extended ICP algorithm computes the registration between pairs of overlapping point sets. Registration of greater than two point sets into a common coordinate frame has been achieved by pairwise registration of pairs of overlapping point sets for example Chen and Medioni [15]. However, pairwise registration does not compute the optimal result. For example if we have three overlapping points sets and we compute the pairwise registration between sets one and two followed by pairwise registration between sets one and three then we do not necessarily minimize the mean square distance between sets two and three.

Bergevin et al. [8] registered multiple range images simultaneously using an extended ICP algorithm to minimize the sum of squared distances for all views. This approach ensures an even distribution of registration errors between overlapping views and achieves errors less than the range image measurement noise for multiple views of complex objects. Eggert et al. [19] performed the registration of multiple overlapping point sets using a force-based optimization. Interconnection between nearest points are represented by springs to give stable convergence to a local minimum for all point sets.

The accuracy of registration obtained using the ICP algorithm depends on the surface shape. If insufficient shape information is available then inaccurate or incorrect registration may occur. Pito [34] presented a registration aid which can be placed in the scene with the object to ensure sufficient shape information for accurate registration of multiple range image views. The surface of the aid is designed such that a range image taken of it from any viewpoint can be accurately registered on a model of the aid.

In the fig. 2.6 a part of a real object consisting of many scanned views is illustrated.

2.3 Surface reconstruction

After scanning of a real object a point set consisting of many thousands or millions points results. Now, the goal is to derive a surface model from the measured points automatically. In the fig. 2.7 an example of reconstructions is illustrated. We are given an unorganized 3D points set **a)** without additional information. For visualization purposes (piecewise polygonal approximation of the surface) a 3D *triangulation* can be built **b)**. Another approach subdivides the whole object's surface into continuous parts, approximates them by analytic / algebraic / CAD etc. surfaces **c)**, in order to create e.g. a CAD computer model, which can directly be manufactured **d)**.

An good overview of existing methods for surface approximation and interpolation from 3D scattered data can be found in [32].

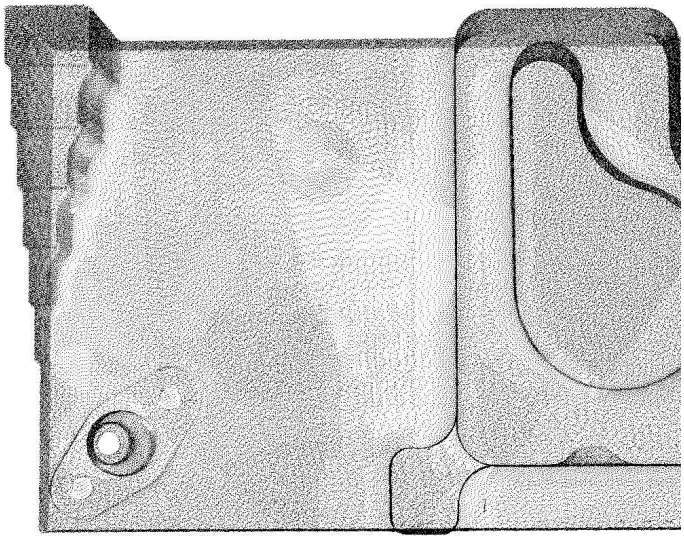
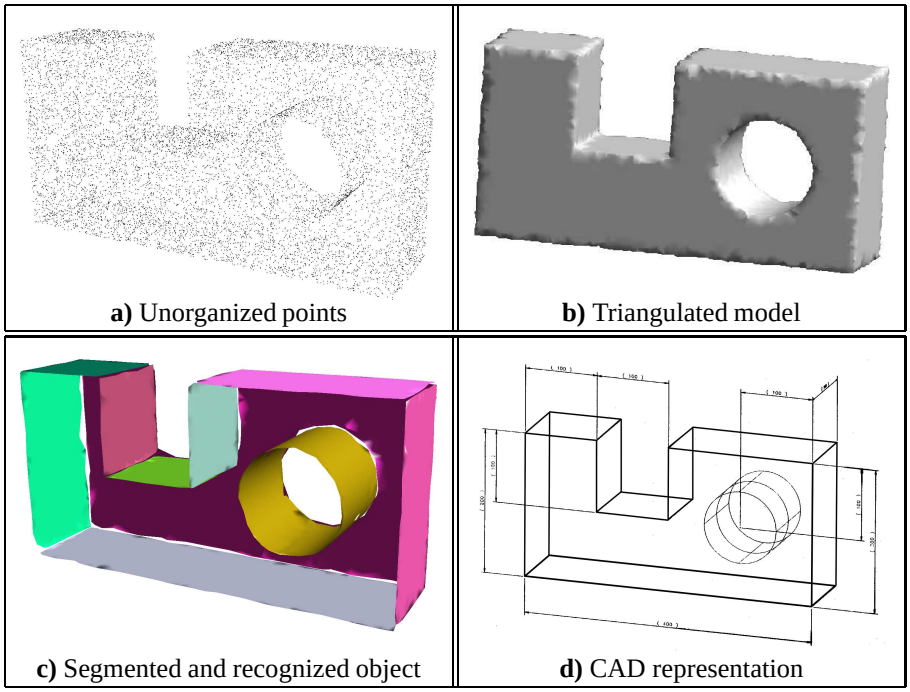


Figure 2.6: Merged views of an scanned real object



a) Unorganized points

b) Triangulated model

c) Segmented and recognized object

d) CAD representation

Figure 2.7: Reverse engineering of an object

2.3.1 Triangulation based on 3D tetrahedrization

One of the fundamental and very often used data structures for the reconstruction is the polygonal approximation of the given points - 3D triangulation. The triangulation is

an attractive representation for computing offsets or boolean operations, for rendering etc. There also exits a huge number of algorithms operating or triangle meshes - mesh reduction methods, subdivision techniques, mesh smoothing, mesh warping etc.

In the past there has been done much work on 3D triangulation, see e.g. [5, 18, 26, 39, 45].

Amenta et al. in [1, 2, 3] presented a new volumetric reconstruction algorithm based on 3D Voronoi diagram. A subset of Voronoi vertices (they called them *poles*) is taken as an approximation of the *medial axis* of the object. To each pole a *polar ball* is assigned and the finite union of the polar balls gives the approximation of the objects surface. Amenta provided a theoretical background to the algorithm and determined the sampling conditions for robust and successful surface triangulation.

Boissonnat [13] based the triangulation algorithm on natural neighbors, which are computed from Voronoi diagram of the sample points. From the Delaunay tetrahedrization the Delaunay facets dual to so-called *bipolar Voronoi edges* are selected as homeomorphic to the surface to be reconstructed. The initial approximation of the surface can be refined by adding additional points on the surface and updating the Delaunay tetrahedrization.

2.3.2 Direct triangulation

Although efficient, robust and fast codes for computing the Voronoi diagram and Delaunay tetrahedrization are now available [17], many researchers try to avoid computing them and base the 3D surface triangulation on creating lower dimensional Delaunay triangulation and *lifting* it to 3D space, see Bernardini et al. [9], Gopi et al. [22], Kós [29] or Heckel et al. [24].

In the approach of Brunnett et at. [14] the neighbor points $\mathcal{N}(P_b) \cup \mathcal{N}(P_e)$ to the edge points P_b and P_e of an edge e are projected into a tangential plane determined by e, a *mate* point among $\mathcal{N}(P_b) \cup \mathcal{N}(P_e)$ is found, merged with the current triangulation sticking the Delaunay criterion and lifted into 3D. The triangulation procedure stores an array of edges of the current triangulation (so-called *frontier*), which are the candidates for the further projection and triangulation. The process is repeated until the frontier is non empty. The results of the triangulation procedure can be seen in the fig. 2.8.

2.3.3 Surface segmentation

Before surfaces can be fitted to the data points, it is necessary to group these points into appropriate subsets, a process which is referred to as segmentation [44, 45]. That segmentation can be based on surface properties, which have been estimated from the data points is an obvious fact. However, to implement an efficient and reliable segmentation is a real challenge.

The aim of this phase of the reconstruction process is to represent the object in the form of a set of surfaces. The object should be cut (subdivided) into various component surfaces, which meet along the boundary curves. In general two different approaches may be considered:

- *edge-based* methods,

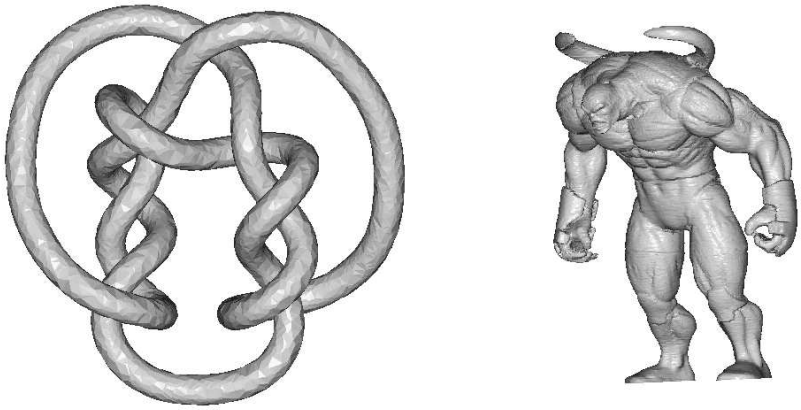


Figure 2.8: Triangulation results

- *face-based* methods.

The *edge-based* method works by trying to find boundaries in the data set, i.e. discontinuities of the normal vectors, principal curvatures or higher derivatives discontinuity along the paths perpendicular to the searched edges. This technique attempts to find the boundary curves and separate the surfaces determined by these edge curves.

The *face-based* method goes in a opposite way, i.e. it collects the points with the same (or very similar) properties and join them into connected regions bounded by edges, which result as an intersections (or other computations) of the surfaces.

Comparison of both methods can be found in [44]: finding the edges in a scanned point set could be very unreliable because of e.g. the specular reflections in the vicinity of edges. Further, in noisy point sets, scanned with insufficient accuracy, the procedure can be confused by noisy data points lying on a regular surface. Filtering or smoothing of the point set do not solve these problems, as the sharp edge are replaced by blends with small radius, which may complicate the edge finding process. The face-based techniques (also region growing techniques) works on a larger number of points. These approaches can also provide a best fit surface to the points and therefore they are often preferred by researchers.

2.3.4 Surface fitting

Surface fitting is very closely bound with the segmentation - it could make sense to consider them as one combined process. Várady and Martin [44] named it “chicken and egg” problem because of the following difficulties:

- If the surface type is known (finished surface fitting), it would be easy to extend the surface by picking the points with a prescribed distance to the known surface (segmentation).

- If a certain portion of the object could be definitely determined as a simple surface (finished segmentation), it would be easy to fit it to a known surface type and obtain its parameters (surface fitting).

Unfortunately, neither of both statements hold. In general, between two techniques for overcoming the mentioned problems may be distinguished:

- *bottom-top* methods,
- *top-bottom* methods.

The bottom-top methods start with a *seed* cluster of points with similar geometric properties, which belong to a known simple surface. Checking the neighborhood of these seed points the surface grows until the "consistency" is met, i.e. the process is stopped, if adding new points would violate the properties of the simple surface.

The top-bottom methods use as initial point cluster the whole data set and attempt to fit them to a simple surface type. If the points are in agreement, the method is done, otherwise the point set is subdivided into subsets and the fitting is recursively repeated to the new subsets.

Several works on surface approximation have been done, e.g. surface approximation using developable surfaces can be found in Chen et at. [16], approximation by ruled surfaces in Pottmann and Chen [35]. Lukács et al. [30] presented a stable methods for approximation of unorganized point by a few simple quadric surfaces (sphere, cylinder, cone, torus).

2.3.5 Surface reconstruction from cross sections

Surface reconstruction from cross sectional contours has become increasingly important in medical image application for visualization of organs, bodies, etc. to the data obtained by imaging techniques as computed tomography (CT), magnetic resonance imaging (MRI) or ultrasonic imaging. These cross sections are the basis for interpolating (or approximating) the boundary surface of the organ.

The problem can be stated as follows: given a series of parallel planar contours, each consisting of a collection of non-crossing, but possibly nested closed simple polygonal curves. The task is to reconstruct a polyhedral solid model, whose intersections along the given planes coincide with the input cross sections. A natural simplification (in contrary to the reconstruction of unorganized data) of this problem is to consider only a pair of successive parallel sections and reconstruct a solid model within the layer delimited by the given parallel planes. The union of these models will give a solution for the full problem.

Fuchs et al. [21] formed the basis of all other subsequent literatures, which said: "Provided two contours of an object which are specified by consecutive points, find out the set of triangular tiles which best approximate the original surface in some sense". Fuchs defined their criterion of the best set as one which maximizes the surface area formed by the set of tiles, using a global optimization technique. The method is well organized, but it is computationally extensive by performing an exhaustive search.

The later researches avoided excessive computational burden by using local heuristics instead of global optimization, e.g. Boissonnat [11, 12] based the reconstruction on 2D Delaunay triangulation of the object contours and extended them into compact 3D tetrahedrization. Using special criteria, the interior tetrahedra or *non-solid connections* are removed from the tetrahedrization resulting the surface of the object. He improved the reconstruction of complex shapes by adding vertices on and inside contours.

The approach of Barequet and Sharir [6] is based on the resemblance between two cross sections. They do not restrict the layout of the contours in one section (if they overlap or not), which was an often condition in many previous works. The similar portions of two adjacent contours are directly triangulated. The remaining parts of the contours are joined into so-called *clefts*, which are thereafter triangulated using the 3D minimum area triangulation technique.

A good overview of many works on reconstruction of 3D object from cross sectional slices is given in Müller et al. in [23].

A part of our research activity is the cooperation with the Anatomic Institute in Göttingen. They dispose of a vast number of sections of various human or animal organs or whole bodies (embryos). The sections are scanned with a microscope, so that a huge image results containing many sectional “skins”. These skins are manually digitized using a tablet and a light pen. Each skin is separated as one data file, see figure 2.9, in order to facilitate the reconstruction process. A visualization of a reconstructed ape embryo is shown in the fig. 2.10.

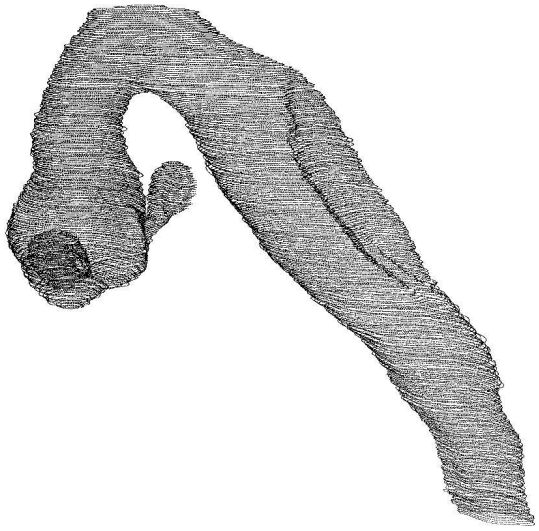


Figure 2.9: Embryo consisting of cross sectional contours - one skin (Courtesy of Anatomic Institute in Göttingen)

Figure 2.10: Reconstructed embryo consisting of three skins and dead cells (Courtesy of Anatomic Institute in Göttingen)

2.4 Definitions

As the input data we obtain n raw points, denoted by P_i, $i = 1,\ldots,n$ - output of the scanning and multiview registration process obtained from an unknown surface Φ. We do not assume any organizing or structure of this data as well as no additional information (as normal vectors or curvatures).

We compute the normal vector in every point P_i (explained in detail in the section 3.2) and denote it as $N(P_i)$. This normal vector $N(P_i)$ together with the point P_i defines a tangent plane $E(P_i)$ of Φ in the point P_i.

Let $c \subset \Phi$ be a regular curve on Φ which results as an intersection of surface Φ with an arbitrary plane $R(P_i) \perp E(P_i)$, where $R(P_i)$ goes through P_i and let $\kappa(c)$ be the curvature of c in P_i. Then κ_n is the normal curvature of Φ in P_i and $\kappa_n = \kappa(c)$.

Obviously there is an infinite number of normal curvatures of Φ in P_i. Among them the minimum and maximum normal curvature $(\kappa_{\min}, \kappa_{\max})$ - the **principal curvatures** - are of great importance.

The principal curvatures on a surface $\mathcal{X}$ define following surface types:

- if $\kappa_{\min} = \kappa_{\max} = 0$, then $\mathcal{X}$ is a part of a plane.
- if $\kappa_{\min} = \kappa_{\max} \neq 0$, then $\mathcal{X}$ is a part of a sphere.
- if $\kappa_{\min} = 0 \wedge \kappa_{\max} = \text{const} \neq 0$, then $\mathcal{X}$ is a part of a cylinder.
- if $\kappa_{\min} = 0$ and $\kappa_{\max}$ increases/decreases linearly along an axis, then $\mathcal{X}$ is a part of a cone.

The principal curvature directions are perpendicular each other.

Principal component analysis (PCA)

Principal component analysis a classical statistical method that involves a mathematical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components. This linear transform has been widely used in data analysis and compression. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.

Traditionally, principal component analysis is performed on a square symmetric matrix of either pure sums of squares and cross products or Covariance (scaled sums of squares and cross products) or Correlation (sums of squares and cross products from standardized data). The analysis results for objects of pure sums and Covariance do not differ. A Correlation object has to be used if the variances of individual variates differ much, or the units of measurement of the individual variates differ. The result of a principal component analysis on such objects will be a new object of type PCA.

The objectives of principal component analysis are

- To discover or to reduce the dimensionality of the data set.
- To identify new meaningful underlying variables.

Principal components are obtained by projecting the multivariate data vectors on the space spanned by the eigenvectors.

The mathematical technique used in PCA is called eigen analysis: we solve for the eigenvalues and eigenvectors of a square symmetric matrix with sums of squares and cross products. The eigenvector associated with the largest eigenvalue has the same direction as the first principal component. The eigenvector associated with the second largest eigenvalue determines the direction of the second principal component. The sum of the eigenvalues equals the trace of the square matrix and the maximum number of eigenvectors equals the number of rows (or columns) of this matrix.

Particularly, PCA is based on the statistical representation of a random variable. Suppose we have a random vector population $\mathbf{x}$, where

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$$

and the mean of that population is denoted by

$$\mathbf{c} = E\{\mathbf{x}\}$$

and the covariance matrix of the same data set is

$$C_x = E\{(\mathbf{x}_i - \mathbf{c})(\mathbf{x}_i - \mathbf{c})^T\}$$

The components of C_x, denoted by c_{ij}, represent the covariances between the random variable components x_i and x_j. The component c_{ii} is the variance of the component x_i. The variance of a component indicates the spread of the component values

around its mean value. If two components x_i and x_j of the data are uncorrelated, their covariance c_{ij} is zero. The covariance matrix is, by definition, always symmetric.

From a sample of vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ we can calculate the sample mean and the sample covariance matrix as the estimates of the mean and the covariance matrix.

From a symmetric matrix such as the covariance matrix, we can calculate an orthogonal basis by finding its eigenvalues λ_i and eigenvectors e_i. The eigenvectors e_i and the corresponding eigenvalues λ_i are the solutions of the equation

$$C_x e_i = \lambda_i e_i, \qquad i = 1, 2, \ldots, m$$

For simplicity we assume that the λ_i are distinct. These values can be found, for example, by finding the solutions of the characteristic equation

$$|C_x - \lambda I| = 0$$

where I is the identity matrix having the same order than C_x. If the data vector has n components, the characteristic equation becomes of order n. This is easy to solve only if n is small. Solving eigenvalues and corresponding eigenvectors is a non-trivial task, and many methods exist. One way to solve the eigenvalue problem is to use a neural solution to the problem. The data is fed as the input, and the network converges to the wanted solution.

By ordering the eigenvectors in the order of descending eigenvalues (largest first), one can create an ordered orthogonal basis with the first eigenvector having the direction of largest variance of the data. In this way, we can find directions in which the data set has the most/the least significant amounts of energy.

Suppose one has a data set of which the sample mean and the covariance matrix have been calculated. Let A be a matrix consisting of eigenvectors of the covariance matrix as the row vectors.

By transforming a data vector $\mathbf{x}$, we get

$$\mathbf{y} = A(\mathbf{x} - \mathbf{c})$$

which is a point in the orthogonal coordinate system defined by the eigenvectors. Components of $\mathbf{y}$ can be seen as the coordinates in the orthogonal base. We can reconstruct the original data vector $\mathbf{x}$ from $\mathbf{y}$ by

$$\mathbf{x} = A^T \mathbf{y} + \mathbf{c}$$

using the property of an orthogonal matrix $A^{-1} = A^T$. The original vector $\mathbf{x}$ was projected on the coordinate axes defined by the orthogonal basis. The original vector was then reconstructed by a linear combination of the orthogonal basis vectors.

Instead of using all the eigenvectors of the covariance matrix, we may represent the data in terms of only a few basis vectors of the orthogonal basis. If we denote the matrix having the K first eigenvectors as rows by A_K, we can create a similar transformation as seen above

$$\mathbf{y} = A_K(\mathbf{x} - \mathbf{c})$$

and

$$\mathbf{x} = A_K^T \mathbf{y} + \mathbf{c}$$

This means that we project the original data vector on the coordinate axes having the dimension K and transforming the vector back by a linear combination of the basis vectors. This minimizes the mean-square error between the data and this representation with given number of eigenvectors.

If the data is concentrated in a linear subspace, this provides a way to compress data without losing much information and simplifying the representation. By picking the eigenvectors having the largest eigenvalues we lose as little information as possible in the mean-square sense. One can e.g. choose a fixed number of eigenvectors and their respective eigenvalues and get a consistent representation, or abstraction of the data. This preserves a varying amount of energy of the original data. Alternatively, we can choose approximately the same amount of energy and a varying amount of eigenvectors and their respective eigenvalues. This would in turn give approximately consistent amount of information in the expense of varying representations with regard to the dimension of the subspace.

Sometimes it is desirable to investigate the behavior of the system under small changes. Assume that this system, or phenomenon is constrained to a n-dimensional manifold and can be approximated with a linear manifold. Suppose one has a small change along one of the coordinate axes in the original coordinate system. If the data from the phenomenon is concentrated in a subspace, we can project this small change δ_x to the approximative subspace built with PCA by projecting δ_x on all the basis vectors in the linear subspace by

$$\delta_y = A_K \delta_x$$

Subspace has then a dimension of K. δ_y represents the change caused by the original small change. This can be transformed back with a change of basis by taking a linear combination of the basis vectors by

$$\delta_x = A_K^T \delta_y$$

Then, we get the typical change in the real-world coordinate system caused by a small change by assuming that the phenomenon constrains the system to have values in the limited subspace only.

Chapter 3

Prerequisites for the segmentation

Any segmentation process is based on geometric properties of the surface to be reconstructed. In this section we will summarize our experiences with different methods to estimate normals and curvatures from the point set. Since the computation of these properties requires the knowledge of the local neighborhood of each surface point, we consider this problem first.

3.1 *k*-nearest neighbors computation

The problem of *k*-nearest neighbors computation within a 3D data set is frequently encountered in Computer Graphics. Applications include e.g. the technique of photon-map rendering where the closest photons to a given one have to be identified, finite elements methods and the segmentation and reconstruction phase within a reverse engineering process, where the surface properties can be estimated from local neighborhood of any point.

In this section we present a new algorithm for *k*-nearest neighbors computation based on median subdivision and a hashing strategy. The major advantage of our hashing function is that bounds can be established that limit the number of points to be inspected during the search process. Estimates for the asymptotic complexity of our search method are given.

The tests have shown, that for points lying on the boundary of an object, our algorithm is always superior to the searching based on a very popular *k*D-Tree data structure [7, 27].

In section 3.1.1 we will introduce the algorithm for organizing the data set. First we perform a recursive subdivision to reduce the complexity of the problem. A median subdivision technique is used in order to create subsets of the same size. The resulting subsets are mapped into hashing tables by our hashing function which can be interpreted as the distance from a particular plane. The main advantage of this hashing

function is that bounds can be established for the number of points to be inspected in the search process.

In section 3.1.2 we present the search algorithm that operates on the data structure resulting from the preprocessing. The basic steps of this methods are the determination of the subsets that have to be inspected and the hashing within this subsets. An analysis regarding memory complexity of our algorithm is presented.

3.1.1 Organizing of the data set

Median determination and subdivision

The 3D points, after scanning from the object's surface, are in general unstructured and they are stored in the data structures in that sequence they were read from the input device. To improve the searching,[1] these points have to be structured. The first step of our organizing is a median subdivision.

The *median* is a value in an arbitrary data set that divides this set in two subsets with the same number of elements (up to one point, if the number of elements is odd).[2]

Median subdivision is more time consuming than *mean* or *interval subdivision*,[3] but it guarantees that the new subsets will contain the same number of points which is essential for our search strategy. Note, that in the case of interval subdivision it can happen, that the number of points in one subset is less than k.

A *histogram* is used for fast determining of the median. Histograms are well known from digital image processing, where they are created by counting the number of pixels for specified values of a certain parameter (e.g. color value or luminance). In our case we count the number of points with coordinate values ranging within a specified interval. Assume that the point set has to be divided with respect to the x coordinate, then the following operations are performed: first we subdivide the interval $[x_{\min}, x_{\max}]$ into r subintervals $I_i,\ i = 1,\ldots,r$. The number of intervals r is chosen depending on the number of points, so that the number of points in one interval is constant on average. Then the number $n(I_i)$ of points per interval is determined. Clearly, if $\sum_{i=1}^{e-1} n(I_i) < \lfloor \frac{n}{2} \rfloor \leq \sum_{i=1}^{e} n(I_i)$, the interval I_e has to contain the median. Finally, the points corresponding to this subinterval are sorted with respect to their x coordinate to determine the median. The worst case is $O(n \log n)$, which occurs if $n-1$ points are lying on a line parallel to x,y or z axis and the last point is lying far from this line. This case occurs very rarely.

Fig. 3.1 illustrates the interval subdivision: the height of a bar over an interval indicates the number of the points with coordinate values within the interval. The critical subinterval I_e, which contains the median, is the interval with 8 elements, which is split below. These elements are sorted and the median is at the position 59.

One subdivision step needs five passes through the point set. The first three passes are needed to determine the median:

[1] Without organizing the k-nearest neighbors to a point can be found in $O(n)$ time

[2] For example, the median is the value in the middle of a sorted data set.

[3] The arithmetic mean of n values can be determined in one pass through the data set.

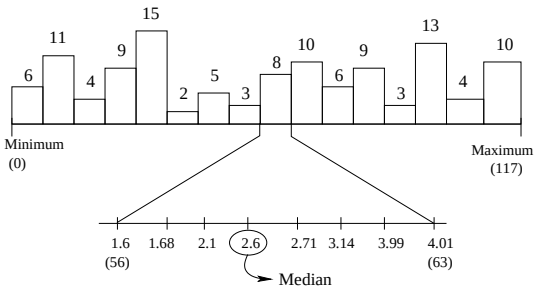


Figure 3.1: Interval histogram and median determining

- The first pass: find minimum and maximum values of the specified coordinate of the point cluster, e.g. $\{x_{\min}, x_{\max}\}$.
- The second pass: create the histogram, i.e. store the number of points per subinterval.
- One pass through the histogram table to determine the subinterval I_e containing the median.
- The third pass: find points that belong to the determined subinterval and sort them (on average the number of points to be sorted can be neglected compared to the number of all points).

After determination of the median the data set must be arranged in such a way, that all points having the specified coordinate less or equal than the median are placed on the left from it in the array and points with a coordinate greater or equal than the median are placed on the right:

- The fourth pass: all medians must be removed from their current position in the array to the middle indices of the array. This step is important if there are many medians in the set.
- The fifth pass: all values less than median are copied to the left from the middle, the values greater than median to the right.

The algorithm has an average case complexity of $O(n)$, which is optimal, because the set must be gone through at least once.

Having determined the median we can perform the median subdivision step. Fig. 3.2 shows the median subdivision of a small point set in 2D. In 3D the process runs analogously.

The point set is subdivided recursively with respect to the x, y and z coordinate such that non overlapping boxes result. After this subdivision a full binary tree is formed, that contains subsets of points in its leaves. In each inner node I of this tree the following data items are stored: the bounding box B of the subtree with parent I, the median value and pointers to the left and right child.

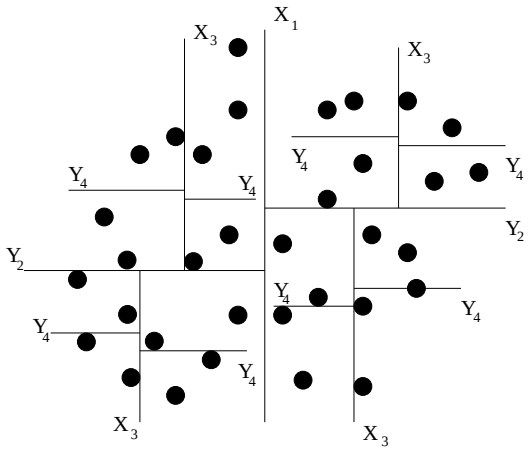


Figure 3.2: Median subdivision in 2D

The exact number of points in each node n_i is not important for our algorithm. We define a threshold $m > k$, which determines the lower bound for the number of points in every subset: $n_i > m > n_i/2$. After subdivision 2^d point clusters result, every cluster contains n_i or $n_i + 1$ points, $n_i \in \{m, m+1, \cdots, 2m\}$; $d = \lceil log_2 \frac{n}{m} \rceil$ is the depth of the binary tree.

Distribution of points in hash tables

The median subdivision creates a binary tree, where all the points are located in the leaves. To speed up the searching procedure we distribute the points in the hash tables. A linear hash function

$$\begin{aligned} \text{Index} &= \left\lfloor \frac{(x+y+z-\text{MinSum}) * \text{TABLESIZE}}{\text{MaxSum} - \text{MinSum}} \right\rfloor, \qquad \text{where} \\ \text{MaxSum} &= \max_i (x_i + y_i + z_i), \\ \text{MinSum} &= \min_i (x_i + y_i + z_i) \end{aligned}$$

is used for the distribution, where 'Index' denotes the index of the point in the hash table, 'MinSum' is the minimum and 'MaxSum' the maximum sum of the coordinates of the points in a point cluster; (x,y,z) are the coordinates of the current point and 'TABLESIZE' is the size of the hash table.

For storage of the points in the hash tables we used an array of direct pointers to arrays of points, see figure 3.3.

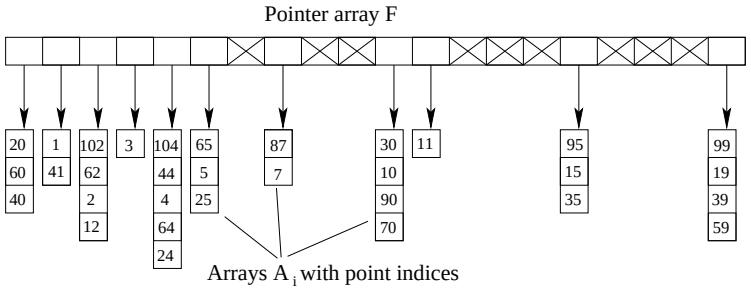


Figure 3.3: Hash table with point lists

3.1.2 The search algorithm

Recall that our data structure is a binary tree, which contains in its leaves the hashing tables with the points, fig. 3.4, and every inner node *I* contains a bounding box *O* of the subtree with root *I.*

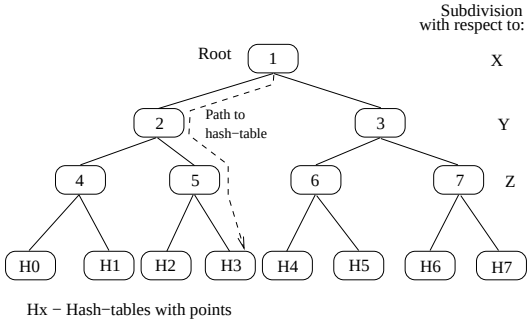


Figure 3.4: Binary tree after median subdivision

Let *P* denote the point for which the *k*-nearest neighbors have to be found. At the beginning of the search process the hash table containing *P* or the adjacent one to the table containing *P* is determined. Note that if *P* contains a coordinate which was used as the median during the subdivision and there were more points with the same coordinate value as the median, the path to *P* is not unique. This fact does not cause any inconveniences if the neighbors to *P* should be determined (in the case we would search for *P*, the subdivision algorithm would have to be changed). The tree parsing scheme

- if the coordinate value of *P* is less or equal to the median, go to the left child,
- otherwise go to the right child

guarantees that the parsing ends in the leaf containing *P* or in the adjacent leaf to the one containing *P*.

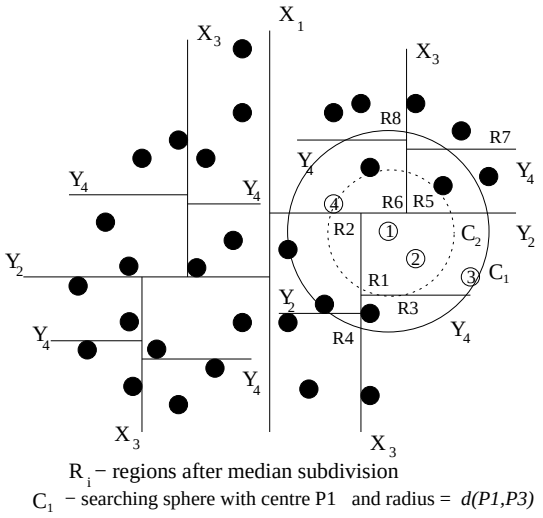


Figure 3.5: 2-nearest neighbors searching in 2D

Then the 'local' k-nearest neighbors of P within that table are found. Fig. 3.5 illustrates for the 2D case that these 'local' nearest neighbors do not have to coincide with the global nearest neighbors of P. Therefore we have to extend our search to boxes in the neighborhood of P.

The overall structure of the recursive algorithm is as follows:
Let $\mathcal{W}$ be a variable, that contains the root of the binary tree. Let $\mathcal{A}$ be an array, where the distances of the found points to P together with these points will be stored. At the beginning we initialize the distances to the maximal possible distance.

Denote the left child of $\mathcal{W}$ with $\mathcal{W}_L$ the right child with $\mathcal{W}_R$, let $\mathcal{B}_s$ be the bounding box of sphere with centre P and radius $d(P,P_k)$, $\mathcal{B}_l$ be the bounding box of a subtree with root $\mathcal{W}_L$ and $\mathcal{B}_r$ be the bounding box of a subtree with root $\mathcal{W}_R$

SearchTree ($\mathcal{W}$) {
  if $\mathcal{W}$ is a leaf of the tree {
    Search in table contained by $\mathcal{W}$
    if a point P_j with $d(P,P_j) < \mathcal{A}[k]$ is found {
      insert P_j at the correct position in $\mathcal{A}$
      $\mathcal{A} = (P_1, P_2, \cdots, P_k)$ is sorted in order
      of increasing distance from P
    }
  }
  else if $P \in \mathcal{W}_L$ {
    SearchTree ($\mathcal{W}_L$);
    if Intersection ($\mathcal{B}_s, \mathcal{B}_r$) is true
      SearchTree ($\mathcal{W}_R$);
  }

```
  else {
    SearchTree (W_R);
    if Intersection (B_s, B_l) is true
      SearchTree (W_L);
  }
}
```

Each time a point Q is found that is closer to P than P_k the following actions are performed, which take $O(k)$ time:

- Insert Q in the sorted neighborhood list: $(P_1, P_2, \cdots, P_i, Q, P_{i+1}, \cdots, P_k)$ with $d(P,P_i) \leq d(P,Q) \leq d(P,P_{i+1})$.
- Remove old P_k from the list, P_{k-1} becomes P_k after insertion.
- Compute the new radius of the searching sphere according to $d(P,P_k)$ and recompute the hash table's interval (see the next paragraph).

We will now describe how the k-nearest neighbors of P within a certain table T are located. Assume that T contains P and let *Pos* be the position of P within T. As an initial set for the k-nearest neighbors we choose the points at neighboring positions to the left and right from P in the hash table. Let P_k be the point with the largest distance from P and let d denote the distance $d(P,P_k)$.

Now we make use of the following fact: if two points $R = (x, y, z)$, $P = (\bar{x}, \bar{y}, \bar{z})$ have a squared distance smaller than d^2, i.e. $(x-\bar{x})^2 + (y-\bar{y})^2 + (z-\bar{z})^2 < d^2$ then the difference of their coordinate sums is smaller than $\sqrt{3}d$: $x+y+z-(\bar{x}+\bar{y}+\bar{z}) < \sqrt{3}d$.

A geometric proof of this fact is illustrated in fig. 3.6 for 2D case. Without loss of generality set $P = (0,0,0)$. Since the expression $\frac{1}{\sqrt{3}}(x+y+z)$ is the distance of the point (x,y,z) from the plane $\pi : x+y+z = 0$ we look for that point inside the sphere S with radius d that has the largest distance from π. Obviously, all points in the interior of S have a distance from π that is smaller than d . Thus $x+y+z < \sqrt{3}d$.

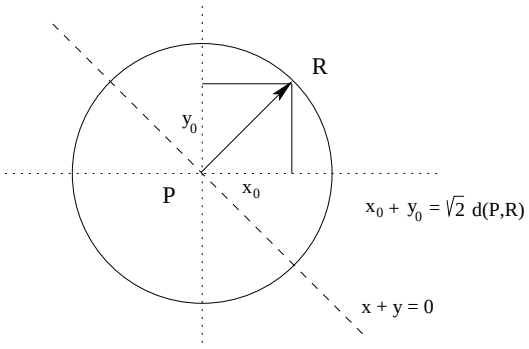


Figure 3.6:

Applied to our problem of nearest neighbor computation we know now that all points closer to P than P_k have indices in the hash table in the interval $\left[Pos-\sqrt{3}\,d, Pos+\sqrt{3}\,d\right]$. Points outside this range do not have to be considered.

Note that by modifying the distance d with εd, $\varepsilon \in [0,1)$ we are able to compute "approximate" k-nearest neighbors [4, 33] and accelerate the searching procedure.

3.1.3 Complexity

Because of two different data structures and algorithms that operate over these structures, it is not trivial to estimate the time complexity and therefore the complexity estimation is partly based on many tests we have done for several objects. The overall time can be divided into two parts: the creation of the data structures (binary tree and distribution of the points in the hash tables) and the search process.

Generation of the data structures

We are given n points. These points are subdivided and for any subinterval a median value must be found (average complexity $O(n)$). In the first step the whole point set must be gone through with $O(n)$ complexity. In the next step the set is subdivided into two subsets each of them with $\frac{n}{2}$ points. For these subsets a median has to be found again with $2O(\frac{n}{2}) = O(n)$ complexity and so on.
In the i_{th} step each of the 2^i subsets must be gone through with $2^i O(\frac{n}{2^i}) = O(n)$ complexity. There are $\lceil \log \frac{n}{m} \rceil$ steps (m is our threshold, see section 3.1.1) therefore the subdivision takes $O(n\log(n))$ time.

In every hash table ($\#\text{Leaves} = 2^{\lceil \log_2 \frac{n}{m} \rceil} - 1$) every point must be projected to the new table. The $\sum_{i=0}^{\#\text{Leaves}} n_i = n$, where n_i is the number of points in the i_{th} hash table. The projection is a linear process, so it takes $O(n)$ time for all tables.

$$\begin{aligned} \text{Creation\_Time} &= O(n\log(n)) + O(n) \\ &= O(n\log(n)) \end{aligned}$$

has an $O(n\log n)$ complexity.

Searching

At the beginning a hash table T_i containing the point P is found. The depth of the tree is: $\text{Depth} = \lceil \log_2 \frac{n}{m} \rceil$. To find that table we traverse from the root to a leaf (*top-down* traversal) and compare the corresponding coordinate of the point with the median value stored in the node that we are going through. This comparison of two floating point numbers takes essentially less time than e.g. determining the distance between two points. The complexity of a traversal in a tree is in general $O(\log n)$.

In the hash table T_i k-nearest points are found. The time for searching in this step strongly depends on the hash function and on the shape of the data subset (in special cases the complexity of hashing can even be linear [38]). If the points are well distributed in the hash table, the position of P can be found in $O(1)$ time. According to the point P_k the bounds are computed, which limit the interval to be looked through. In

the worst case all points must be checked but the complexity stays $O(1)$ because of the constant number of points per hash table. The number of points which do not have to be checked (points outside of the computed interval), cannot be determined, it depends only on the radius of the searching sphere. Many tests have shown, that if we omit these bounds, the searching takes up to 3 times longer.

Thereafter, on the back trace to the root, all neighbor hash tables are checked, if their bounding box intersects with the searching sphere. Again the tests have shown, that the number of checked points does not depend on the sampling depth. Suppose we have an object, that was scanned many times with different sampling depths. The higher the number of points, the nearer are the adjacent points, which causes smaller searching spheres. The number of checked points only depends on the shape of the object (i.e. distribution of the points in the hash tables) and on k. Thus, we can assume that for a constant sampling the number of checked hash tables stays constant and therefore the complexity for hash table searching is in average $O(1)$.

The worst case for searching in hash tables would be a distribution of the points, which belong to one hash table, exactly on a plane parallel to the plane $x+y+z=0$ (as the hash function). In this case the searching complexity would degenerate to linear time. This can happen for artificially generated data. But we work with scanned object surfaces and we assume that no 3d scanner works with infinite precision, and therefore the points are always contaminated with some noise. The probability that these noisy points would lie exaclty on a plane is very small. In the case they do not lie on that plane, the minimum and maximum sum of the points' coordinates are not equal and the coordinates range can be mapped on the hash table, as shown in the section 3.1.1.

So the searching complexity can be expressed as: Searching $\sim O(\log n)$. The searching time depends on k, but for a constant k it can be excluded from the $O()$-notation.

3.1.4 Memory complexity

Two data structures are stored in memory: binary tree and hash tables. Let N_n denote the number of nodes and N_l the number of leaves. As we created a full binary tree it stands: $N_n = N_l - 1$.

There are

$$\begin{aligned} N_n + N_l &= (N_l - 1) + N_n \approx 2N_l = \\ &= 2^{\lceil \log_2 \frac{n}{m} \rceil + 1} \approx 2\frac{n}{m} \end{aligned}$$

structures which denote the nodes of the binary tree. So the memory complexity for the binary tree is $O(n)$.

The hash table takes n pointers and counters (pointer array) and n points (point arrays), which take together $12n$ bytes, so $O(n)$.

The whole memory complexity is $O(n) + O(n) = O(n)$.

3.1.5 Time comparison

In this section we present performance results of our method. As test examples we used a data set sampled randomly from a torus and a box surface and several digitized

objects.

All times for searching are in seconds and the tests were performed on a PC with Athlon 1200 MHz processor and 512 MB DDR-RAM. The whole time is divided in two parts:

- *Preprocessing time*: organizing of the data structures, that is done only once for the data set
- *Searching time*: includes the searching for k-nearest neighbors to each point of the data set.

Points	**Prep.**	**Searching for # of points**					
		1	10	15	20	25	30
10K	0.05	0.08	0.38	0.50	0.62	0.80	0.91
50K	0.37	0.32	1.55	2.16	2.78	3.43	4.01
100K	0.91	0.66	3.08	4.32	5.52	6.78	8.05
250K	2.41	1.72	8.18	11.42	14.66	17.82	21.20
500K	6.20	3.63	17.14	24.26	31.03	38.13	45.40
1M	13.10	7.54	35.50	49.80	64.11	78.73	94.03
2M	39.48	15.74	72.73	102.59	132.31	163.57	195.88

Table 3.1: Searching times for median-hashing search algorithms

3.2 Approximation of the normal vectors

In the following we will assume that the set $\mathcal{N}(P)$ of k-nearest neighbors of a point P computed with respect to the Euclidean distance is the same as the set of the k-nearest neighbors of P with respect to geodesic distance, i.e. on the surface of the initial object.

Note, that it is this assumption that allows to estimate higher order surface properties as normal vectors or principal curvatures from the data set.

A very common approach to approximate the normal in P is to compute the plane of regression R (see [20]) of the data set $N(P) \cup \{P\}$ and to use the normal vector of R as the approximation.

This method is fast and easy to implement but it approximates the normal vectors of analytic surfaces (or quadric surfaces) very roughly. If the object consists of many smooth curved surfaces, this method does not provide a satisfactory estimation of the normal vectors.

In this section we present three different methods for normal vector estimation - sections 3.2.1, 3.2.2, 3.2.3 and discuss their stability and efficiency - section 3.3.

3.2.1 Local Centre Triangulation (LCT)

The main idea of this method is that the centre C_m of the local neighborhood (denoted with $P_1, P_2, \ldots, P_k$) of P lies close to P. So, if we estimate the normal vector in C_m, it will be close to the normal vector of P, see fig. 3.7.

The algorithm works as follows:

- Compute the centre C_m of the neighborhood of P ($C_m = \frac{1}{k}\sum_{i=1}^{k} P_i$)
- Triangulate the neighborhood according to C_m:
  1. Find an appropriate projection plane E as described below
  2. Project the points P_i to this plane and store their 2D coordinates as P_i^{2D}
  3. Sort points P_i^{2D} according to their polar coordinates starting with the line segment $\vec{C_m P_1}$ as X-axis. Denote these points with $P_j^{2D}; j = 1,2,\ldots,k$.
  4. Build a triangulation $\mathcal{T}_2$ in 2D from P_j^{2D}: a triangle T_j^{2D} is built from $(P_j^{2D}, C_m, P^{2D}_{(j \bmod k)+1})$, $j = 1,\ldots,k$
  5. Build a triangulation $\mathcal{T}_3$ in 3D from $\mathcal{T}_2$ (each point P_j^{2D} corresponds to a point P_i in 3D)
- Compute the normal vector $\vec{N}_i$ in every 3D triangle
- Set the normal vector $\vec{N}$ in C_m to be the average $\vec{N} = \frac{1}{k}\sum_{i=1}^{k} \vec{N}_i$, note that k triangles were created, see 4)
- Use $\vec{N}$ as the estimated normal vector in P

In the algorithm a plane E was needed for the projection of the points into 2D. We choose E from the set of planes containing P_1, P_2 and a third point taken from $P_i, i = 3,\cdots,k$, so that the sum of the orthogonal distances of all P_i to E is minimal.

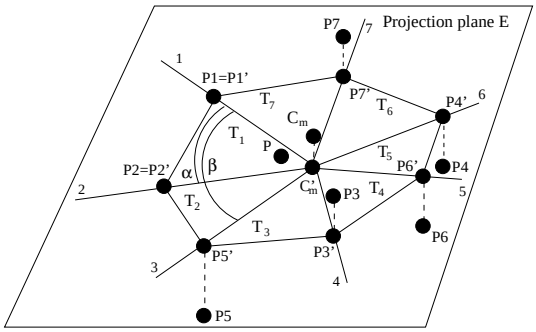


Figure 3.7: Local Centre Triangulation

The complexity of the estimation process for all points is $O(k\log(k).n)$ (sorting of P_i^{2D}), where k is the number of neighbors, n is number of points.

3.2.2 Local Delaunay Triangulation (LDT)

In this method we triangulate the neighborhood of P using Delaunay triangulation. The estimated normal vector is the average of the normal vectors of the triangles incident with P, see fig. 3.8.

Algorithm:

- Find the best projection plane E containing P and P_1 (see the next paragraph)
- Project points P_i to this plane and store their 2D coordinates as P_i^{2D}
- Use Delaunay triangulation for the 2D points P_i^{2D} and thereafter build a 3D triangulation from the points P_i, that correspond to P_i^{2D}.
- Compute the average of the normal vectors of the triangles incident with P

Like the LCT, Delaunay triangulation also needs a projection plane E. For the determination of E we use a similar method as described above, which differs only in the fact, that P is included in the neighborhood, therefore E contains P.

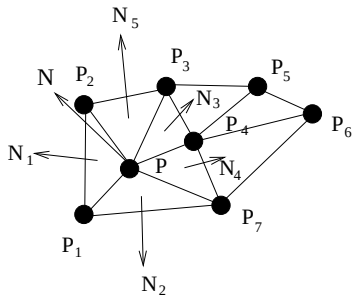


Figure 3.8: Local Delaunay Triangulation

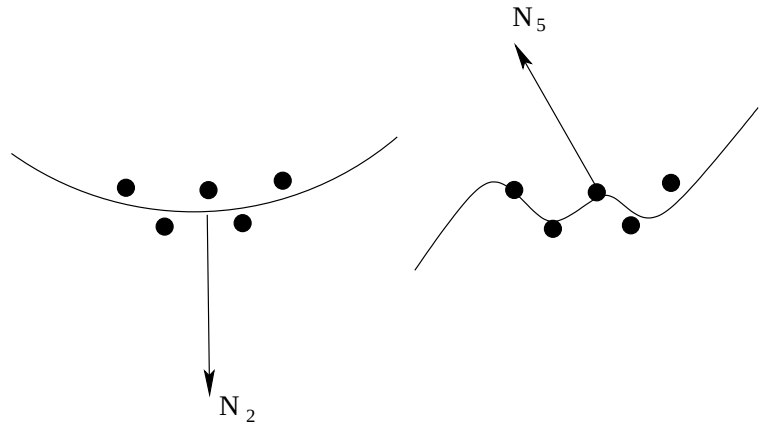
The complexity of this normal estimation is $O(k^2.n)$,[4] (k is the number of the neighbors).

3.2.3 Approximation with an analytic surface (AwAS)

The task of this method is to find the best quadratic or cubic function, that fits the neighboring points of P. Quadratic and cubic surfaces can well approximate the common types of a small point set. Surface of higher degree would approximate a noisy point set "too well", i.e. there could occur unwanted oscillations (see Fig. 3.9), which can lead to bad approximation values.

As illustrated in fig. 3.10 the result of the approximation strongly depends on the choice of coordinate system, which is used for the determination of the best fitting polynomial.

[4] For the building of the Delaunay triangulation we used an incremental method, which is not optimal. The optimal complexity is $O(k \log k.n)$



Approximation with a quadratic curve Approximation with a quintic curve

Figure 3.9: Comparison of a quadratic and quintic approximation

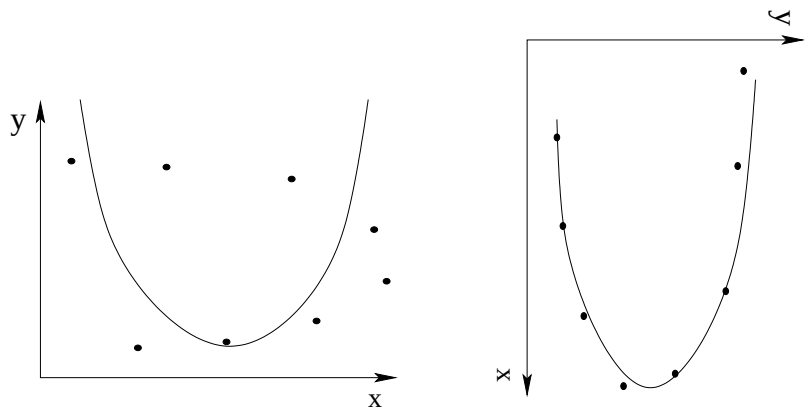


Figure 3.10: Approximation of points with a quadratic curve in the original and rotated coordinate system

To determine an appropriate coordinate system we compute a first estimation of the normal vector $\vec{N}_f$ in P using either best fit plane or LCT procedure. Then we define a new coordinates system by coinciding $\vec{N}_f$ with the z-axis, origin is at P. This new coordinate system is used for the computation of the best fitting polynomial.

The Algorithm:

- Estimate the normal $\vec{N}_f$ in P using plane of regression or LCT. Define a new coordinate system: $z \to \vec{N}_f, \text{Origin} \to P$.
- Transform P and its neighbors $P_i, i = 1, \ldots, k$ to the new coordinate system. We denote these points as $P_i^t, i = 0, \ldots, k$; $P_0^t = P^t$.

- Approximate P_i^t with a quadratic or cubic function ($z = f(x,y)$) using the least square method, e.g. for the quadratic surface: $\sum_{i=0}^{k}(Ax_i^2 + Bx_iy_i + Cy_i^2 + Dx_i + Ey_i + F - z)^2 \rightarrow \min$; where $[x_i, y_i, z_i]$ are the coordinates of P_i and $A, \cdots, F$ are the coefficients to be found.
- $\frac{\partial f(x,y)}{\partial A} = 0, \cdots, \frac{\partial f(x,y)}{\partial F} = 0$ leads to a system of 10 for cubic, resp. 6 for quadratic, linear equations, which can be be solved easily.
- Compute a normal vector $\vec{N}_s$ as $(1, 0, \frac{\partial f(x,y)}{\partial x}) \times (0, 1, \frac{\partial f(x,y)}{\partial y})$.
- Transform the normal vector $\vec{N}_s$ to the original coordinate system.

The *approximation with an analytic function* gives for smooth surfaces almost exact results and works very fast. The weakness of this method are objects with bad or insufficient sampling: fig. 3.11 demonstrates such a case (in 2D), when the six neighbors of P (the whole neighborhood) are on its left side. Due to the insufficient sampling and small irregular neighborhood of P the cubic function provides rather worse estimation of the normal vector as e.g. the plane of regression would do. One solution to overcome this problem is an adaptive (automatic) extension of the number of the neighbors, but these cases are very time consuming to detect.

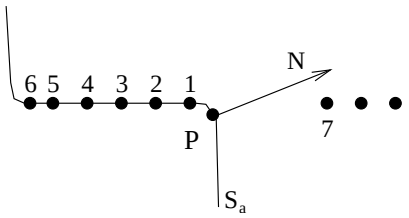


Figure 3.11: Approximation with a cubic curve of a surface with sampling flaws

3.3 Comparison of the methods for normal vector estimation

We performed numerous tests on artificially created data, where the exact normal vector could be computed. We contaminated these data sets with random noise, in order to check the stability of the mentioned methods. In tables 3.2, 3.3 and 3.4 we show in every cell of the table the average and the maximum angle (in degrees) between the exact and the estimated normal vector and the third value is the computation time.

For artificial data set without noise the LDT estimates the normal vectors very well, even for normal vectors of the points close to the edges. However, for noisy data sets this method showed stability problems. A further disadvantage results from the fact that the complexity increases fast with the size of the neighborhoods.

The LCT method works much faster than the LDT but provides normals that are less exact. On the other hand, it works more stable than the LDT for noisy data sets.

The AwAS method estimates the normal vectors perfectly, if they belong to a smooth surface without noise. Normal vectors close to the edges are estimated worse than with LCT or LDT. On the other side, for noisy point sets it works more stable and up to 7times faster than the LCT.

For all four methods (best fit plane, LCT, LDT, AwAS) the choice of the neighborhood size is very important. Our tests have shown, that the appropriate neighborhood size for point sets with noiseless sampling is in the range of $k = [8, 12]$. For this neighborhood size the estimation was incorrect only on sharp edges. Increasing of the neighborhood size results in a smoothing effect of the normal vectors in the vicinity of the edges, but did not improve them on smooth surfaces. Note, that using 14 instead of 10 neighbors the results get worse for objects with edges (C^0 transitions), because of smoothing close to the edges, see tables 3.2, 3.3.

For noisy data sets the neighborhood size has to be increased in order to obtain satisfactory results. However, these large neighborhoods result in a strong smoothing close to the edges. Extensive tests showed, that the optimal choice of the estimation method should be made depending on the surface shape. In general the most reliable method for badly scanned objects seems to be the combination of best fit plane with AwAS, where the neighborhood size should be chosen in the range of 15–25 neighbors. For very noisy point sets up to 30 neighbors has to be chosen and the best fit plane method seems to be the most stable choice.

In our implementation the user provides one important parameter at the beginning of the whole reconstruction process. This parameters denotes the user's rough estimation of the noise in the point set. The user only determines its subjective approximation in the range of $[0, 100]$ (e.g. 0 = noiseless, 50 = medium noise, 100 = very noisy data). According to this parameter the normal vector estimation method is chosen (e.g. up to 70 we compute the normal vectors with AwAS, otherwise we use the BFP method) and the neighborhood size computed. This value is also used for determining the neighborhood size for principal curvatures computation (section 3.5), for the estimation of the segmentation parameters of the second order segmentation (section 4.3) and for the final adaptation of the postprocessing parameters to the data (chapter 5). Note that our implementation also allows to specify all the parameters manually, resp. to alter the offered parameters.

3.4 Consistent orientation of the normal vectors

The subsequent segmentation step, see chapter 4, needs consistent oriented normal vectors. For this we employ the orientation propagation method suggested by Hoppe [25]. The direction of one normal vector is chosen and this direction is propagated to adjacent points traversing the weighted minimal spanning tree of the points, where the weights correspond to the angles between normal vectors of adjacent points. Note that this method works satisfactory for well and average quality sampled data, but our tests have shown that it may fail for highly complex data sets. In this case it is necessary to allow different neighborhood sizes for different parts of the point set.

If the object possesses sharp edges this approach may fail as illustrated in fig. 3.12. Part a) of the figure shows identical normal vectors in different situations, where it is

Noiseless data set: 10 neighbors				
	BFP	LCT	LDT	AwAS
10*k*	3.384 / 89.05 / 0.15	3.261 / 90 / 1.39	2.657 / 89.4 / 5.12	**3.205 / 87.5 / 0.41**
100*k*	1.121 / 89.8 / 1.15	1.068 / 90 / 13.59	0.845 / 89.9 / 51.72	**0.993 / 89.8 / 3.8**
Noiseless data set: 14 neighbors				
10*k*	3.932 / 79.17 / 0.2	3.671 / 86 / 2.47	2.917 / 89 / 8.13	**3.556 / 75.9 / 0.51**
100*k*	1.277 / 87 / 1.46	1.202 / 89 / 24.04	0.945 / 89.5 / 93.9	**1.169 / 87 / 4.41**
Noise: 100%, 0.1% of a bounding box's diagonal: 15 neighbors				
10*k*	4.34 / 79.6 / 0.29	4.247 / 88.9 / 2.71	4.852 / 90 / 10.40	**4.134 / 86 / 0.63**
100*k*	**2.461 / 86 / 2.62**	2.866 / 89.7 / 27.16	6.723 / 89 / 103.74	2.719 / 83.56 / 6.15
Noise: 100%, 0.1% of a bounding box's diagonal: 20 neighbors				
10*k*	4.763 / 72 / 0.32	4.598 / 89.8 / 4.55	5.074 / 89.9 / 16.04	**4.534 / 81 / 0.81**
100*k*	**2.302 / 82 / 3.11**	2.662 / 89 / 45.22	6.669 / 89.7 / 156.8	2.388 / 71.8 / 7.26
Noise: 100%, 0.3% of a bounding box's diagonal: 15 neighbors				
10*k*	5.046 / 89.7 / 0.31	5.223 / 89 / 2.76	8.163 / 89.7 / 10.23	**5.01 / 70.43 / 0.66**
100*k*	**4.856 / 87.7 / 2.56**	6.315 / 89.7 / 27.17	15.54 / 90 / 102.88	5.814 / 87.74 / 5.89
Noise: 100%, 0.3% of a bounding box's diagonal: 20 neighbors				
10*k*	5.363 / 79.8 / 0.36	5.423 / 84.6 / 4.56	8.23 / 89.7 / 15.60	**5.059 / 71.11 / 0.76**
100*k*	**4.03 / 86.11 / 3.10**	5.344 / 89.8 / 45.36	15.15 / 90 / 153.48	4.377 / 86.46 / 6.99
Noise: 100%, 0.3% of a bounding box's diagonal: 25 neighbors				
10*k*	5.434 / 75 / 0.38			**5.421 / 77.04 / 0.80**
100*K*	**3.598 / 83 / 3.61**			3.792 / 75.9 / 8.09

Table 3.2: Normal estimation comparison for a box

impossible (based on the available information) to detect which case is being processed. Part b) shows opposite normal vectors on the same edge in a situation where it is again impossible to orient consistently without additional information.

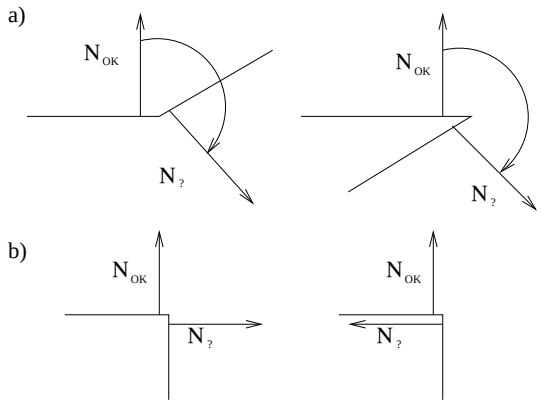


Figure 3.12: Normal vector orientations problems

Noiseless data set: 10 neighbors				
	BFP	LCT	LDT	AwAS
10k	2.289 / 90 / 0.23	2.087 / 89.6 / 1.41	1.793 / 88.6 / 6.26	**1.282 / 89 / 0.50**
100k	0.722 / 90 / 1.78	0.6525 / 89.67 / 13.59	0.6056 / 89.7 / 59.81	**0.4066 / 89.8 / 4.39**
Noiseless data set: 14 neighbors				
10k	2.252 / 89.2 / 0.28	2.085 / 88.9 / 2.43	2.001 /89.8 / 9.78	**1.507 / 89.8 / 0.55**
100k	0.757 / 90 / 2.15	0.671 / 90 / 24.02	0.662 / 90 / 96.12	**0.45 / 89.6 / 5.15**
Noise: 100%, 0.1% of a bounding box's diagonal: 15 neighbors				
10k	2.872 / 88.9 / 0.3	3.00 / 89.7 / 2.78	5.572 / 89.7 / 10.70	**2.679 / 90 / 0.62**
100k	**3.31 / 90 / 2.59**	4.385 / 90 / 27.04	11.26 / 89.9 / 107.15	3.963 / 90 / 5.86
Noise: 100%, 0.1% of a bounding box's diagonal: 20 neighbors				
10k	2.84 / 89.3 / 0.34	2.926 / 89.5 / 4.56	5.819 / 89.6 / 16.28	**2.55 / 89.9 0.79**
100k	**2.644 / 89.9 / 3.02**	3.631 / 90 / 45.15	11.12 / 90 / 163.17	2.907 / 89.9 / 6.98
Noise: 100%, 0.3% of a bounding box's diagonal: 15 neighbors				
10k	**4.524 / 89.9 / 0.30**	5.343 / 89.7 / 2.73	11.77 / 90 / 10.70	4.905 / 89 / 0.60
100k	**9.56 / 90 / 2.57**	13.25 / 89.9 / 27.08	26.67 / 90 / 104.38	10.97 / 90 / 5.88
Noise: 100%, 0.3% of a bounding box's diagonal: 20 neighbors				
10k	4.12 / 89.9 / 0.34	4.904 / 90 / 4.56	11.63 / 90 / 15.94	**4.035 / 89 / 0.72**
100k	**7.148 / 89.9 / 3.05**	10.6 / 90 / 45.20	26.41 / 90 / 156.21	7.796 / 90 / 6.82

Table 3.3: Normal estimation comparison for a cone

To overcome this problem the neighborhood may be increased to smoothen the normal vectors close to the sharp edges (which can result into worse normal vector estimation). In the figure 3.13 the neighborhood size problem is illustrated on a data set: the figure a) shows the whole point set; figure b) shows the point set with normal vectors after orientation propagation procedure, where $k = 10$ neighbors were taken. It is obvious that the smoothing of the normal vectors on the edge of the cone was insufficient to guarantee correct propagation over this edge. In the figure c) $k = 20$ neighbors provided sufficient smoothing for correct orientation.

In the figure 3.13 an other problem can be demonstrated: the gap between the letters (i.e. between two objects of a scene or between two separate parts of an object) is bigger than the maximum distance of a point P to its k_{th} neighbor. In particular this means that during parsing the neighborhood of the points the procedure is not able to continue on the next object (in this case on the adjacent letter). If the orientation procedure detects, that all points of an object were parsed, but there left some points unmarked, it assumes that the scene consists of several objects and the procedure proceeds with searching for the next non-marked point with the smallest z coordinate (as proposed Hoppe in the single object case). If the scene consists of many *closed* objects, our orientation procedure is able to orient all objects consistently (depending on the object's sharp edges and the neighborhood size). If the objects contain boundaries, a consistent orientation of the whole scene cannot be determined automatically (the user can invert the orientation of a particular object interactively).

Using the precomputed k neighbors $\mathcal{N}(P)$in every point P, its normal vector $N(P)$ and a simple hash algorithm, we are able to perform the orientation procedure in $O(kn)$

Noiseless data set: 10 neighbors		
#Points	BFP	AwAS
10*k*	3.384 / 89.05 / 0.15	**3.205 / 87.5 / 0.41**
100*k*	1.121 / 89.8 / 1.15	**0.993 / 89.8 / 3.8**
Noiseless data set: 14 neighbors		
10*k*	3.932 / 79.17 / 0.2	**3.556 / 75.9 / 0.51**
100*k*	1.277 / 87 / 1.46	**1.169 / 87 / 4.41**
Noise: 100%, 0.1% of a bounding box's diagonal: 15 neighbors		
10*k*	1.409 / 5.7 / 0.27	**0.7374 / 6.58 / 0.62**
100*k*	**1.818 / 10.71 / 2.52**	2.33 / 29.26 / 5.82
Noise: 100%, 0.1% of a bounding box's diagonal: 20 neighbors		
10*k*	1.303 / 5.91 / 0.34	**0.5043 / 3.52 / 0.75**
100*k*	**1.357 / 10.85 / 3.01**	1.537 / 21.23 / 6.8
Noise: 100%, 0.3% of a bounding box's diagonal: 15 neighbors		
10*k*	**5.677 / 88.1 / 2.56**	7.147 / 89.54 / 5.75
100*k*	**3.891 / 28.25 / 1.33**	4.973 / 51.24 / 3.11
Noise: 100%, 0.3% of a bounding box's diagonal: 20 neighbors		
10*k*	**2.86 / 16.16 / 1.59**	3.291 / 25.65 / 2.56
100*k*	**4.074 / 25.43 / 3.05**	4.726 / 39.94 / 7.13
Noise: 100%, 0.3% of a bounding box's diagonal: 25 neighbors		
10*k*	**2.287 / 11.26 / 1.82**	2.493 / 17.12 / 3.96
100*K*	**3.189 / 18.11 / 3.56**	3.545 / 30.82 / 8.03

Table 3.4: Normal estimation comparison for a torus

time. Our algorithm looks as follows:

1. Unmark all points.
2. Find a non-marked point P_z with the smallest z coordinate and call procedure `Insert` (P_z)[5] . If no such point exits, finalize the procedure.
3. Take the first point P (with its neighbor P_j) from the hash table and delete it from the hash table.
4. If the hash table is empty, go to 2.
5. If P_j is marked, continue with 3.
6. Otherwise orient P_j according to P, mark it and insert both points into the hash table (if possible) - `Insert` (P), `Insert` (P_j).
7. Continue with 3.

`Insert` (P)

[5]The function `Insert` (P) inserts a point P into a hash table, see the next paragraph

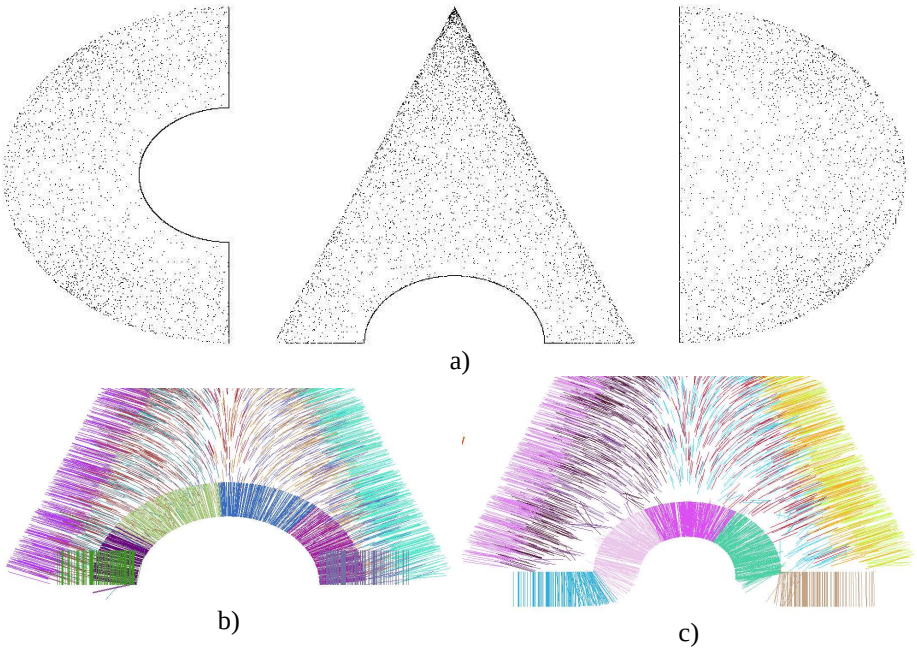


Figure 3.13: Neighborhood depending orientation propagation

- Find in $\mathcal{N}(P)$ a non-marked neighbor P_j with the smallest angle to $N(P)$, i.e. $\gamma = \min_j (\angle(N(P_j), N(P)))$.
- If no such point exists, return directly to the main procedure.
- Otherwise $\gamma = \min(\pi - \gamma, \gamma)$ and project P according to γ to the hash table: $I = \left\lfloor \frac{2\gamma}{\pi} * \text{Tablesize} \right\rfloor$.
- Insert the point pair $[P, P_j]$ as a new head item of the linked list of point structures at index I of the hash table.

The `Insert` procedure finds in the neighborhood of P its neighbor with the smallest angle γ to P and projects according to γ the pair $[P, P_j]$ into the hash table. The procedure consists of only one loop through all neighbors of P, so its complexity is $O(k)$.

The main procedure has to visit every point at least once. Depending on the shape of the object can happen that some points are inserted into the hash table many times (at most k times) and other points are inserted even only once, because of all their neighbors has already been oriented correctly.

We separate k points: the worst case occurs if the first point P_1 is inserted into the hash table k times, what causes correct orientation for the remaining $k-1$ points, so that they have to be inserted into the hash table only once. The average number of

insert operations is:

$$\frac{k+(k-1)}{k} = \frac{2k-1}{k} \approx 2$$

The worst case complexity for our orientation algorithm is $O(2kn)$, and so the overall complexity is $O(kn)$.

Note that one small inaccuracy has to be mentioned: on one index of the hash table a few point items can be linked. We do not search for the smallest one, because the error that we do, can be neglected: let γ_i, γ_j be the smallest angles corresponding to an index I and its next index $I+1$. We obtain $I = \left\lfloor \frac{2\gamma_i}{\pi} * \text{Tablesize} \right\rfloor = \frac{2\gamma_i}{\pi} * \text{Tablesize}$ and $I+1 = \left\lfloor \frac{2\gamma_j}{\pi} * \text{Tablesize} \right\rfloor = \frac{2\gamma_j}{\pi} * \text{Tablesize}$. Then $\varepsilon = \gamma_j - \gamma_i = \frac{\pi}{2\text{Tablesize}}$ is the maximum error we do, if we take the first point item from an index I of the hash table and do not search for the smallest one on the index I. We set Tablesize = number of points, or minimal 10,000. So $\varepsilon < 0.0180°$.

Computation time of a few real objects can be seen in the table 3.5.

Points set	#points	Time	Neighborhood	Type
CAD	15,999	0.24	20	closed, many objects
Tank	27,885	0.31	12	open, single object
Picard	73,014	0.76	12	open, single object
Mocca	193,856	2.77	20	closed, single object
Keiper	294,423	3.38	15	open, many objects

Table 3.5: Normal vector orientation times

3.5 Computation of principal curvatures

For second order segmentation the knowledge of the principal curvatures is of fundamental importance. To compute them we use the approximation method suggested by Martin et al. [31]. We take the computed normal vector N_p at a point P and fix a new coordinate system with the origin at P and z-axis pointing in the direction of the normal vector N_p. In this coordinate system we approximate the neighbor points of P with a quadratic surface $f(x,y) = ax^2 + bxy + cy^2 + dx + ey + f$ using the least square method $\sum_{i=0}^{k}(f(x_i,y_i) - z_i)^2 \rightarrow min$ [6]. The normal vector in the modified coordinate system has than the form $N_p^m = (-\frac{\partial f}{\partial x}, -\frac{\partial f}{\partial y}, 1)$. After computation of the coefficients g_{ij} and h_{ij} of the first and second fundamental form, we obtain the principal curvatures as the solution of the quadratic equation $det(h_{ij} - \lambda g_{ij}) = 0$, i.e.

$$\begin{vmatrix} \frac{\partial^2 f}{\partial x^2} - \lambda\left(1 + \left(\frac{\partial f}{\partial x}\right)^2\right) & \frac{\partial^2 f}{\partial x \partial y} - \lambda\left(\frac{\partial f}{\partial x}\right)\left(\frac{\partial f}{\partial y}\right) \\ \frac{\partial^2 f}{\partial x \partial y} - \lambda\left(\frac{\partial f}{\partial x}\right)\left(\frac{\partial f}{\partial y}\right) & \frac{\partial^2 f}{\partial y^2} - \lambda\left(1 + \left(\frac{\partial f}{\partial y}\right)^2\right) \end{vmatrix} = 0$$

[6] We also tested the approximation with a cubic surface, but the quadratic surface turned out as more stable for noisy data.

The accuracy of the principal curvatures estimated by this method obviously depends on the quality of the approximated normal vectors. Furthermore, the curvatures depend on the size of the chosen neighborhood. In general, the neighborhood for curvature estimation has to be larger than the one for normal estimation. The neighborhood has to be enlarged with the increase of noise contained in the data. As introduced in section 3.3 we again use the noise parameter to alter the neighborhood size depending on the user's rough estimation of the noise in the data set. The neighborhood size varies in the range $k = 20$–40.

Chapter 4

Segmentation

The relevance of the first order segmentation is to detect sharp edges, regions of high curvature as well as flat areas in the object to be reconstructed. Since for a reliable curvature estimation sharp edges have to be located, first order segmentation is a prerequisite for higher order segmentation.

One of the main features of our method is to proceed by alternating the steps of segmentation and normal estimation as described below. Each segmentation allows to improve the normal estimation which in turn can be the basis for a refined segmentation.

4.1 Segmentation of the surface based on normal vectors

After estimation of the normal vectors the segmentation of the object's surface can be started. This normal vector based segmentation tries to find tangent discontinuities of the surface, i.e. big differences between adjacent normal vectors. Therefore a prescribed input angle α is used to specify the maximum acceptable angle between the adjacent normal vectors.

We start the segmentation with an arbitrary point P_a and its normal $\vec{N}_a$, which builds a starting cluster C_a and test the angle between $\vec{N}_a$ and the normal vectors of all its neighbors. If the angle between $\vec{N}_a$ and $\vec{N}_n$, a normal vector of a point P_n from the neighborhood of P_a, is smaller than α, P_n will be added to the cluster C_a and after checking all neighborhood points the procedure proceeds recursively with P_n.

This process has the following disadvantages: if the object consists of several surfaces without sharp edges (e.g. torus, sphere etc.), the whole object would be recognized as one segment, which would be very hard to approximate directly with one surface in the later step. Furthermore, we cannot assume, that the normal vectors are exact, e.g. if the normal vectors were not supplied with the point set but had to be estimated. The described methods cannot estimate the normal vectors correctly in the vicinity of sharp edges, but they will provide a smooth transition of the normal across

a sharp edge (see fig. 4.1). Thus, the segmentation process would not recognize some sharp edges and could declare two perpendicular planes as one segment.

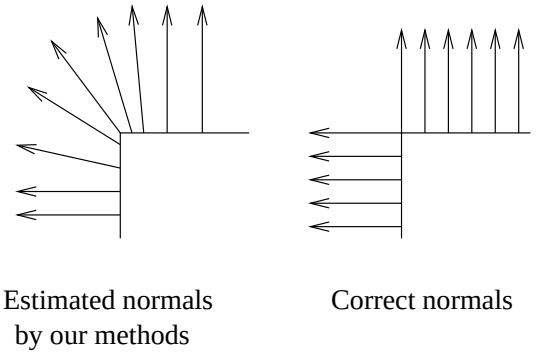


Figure 4.1: Correct and estimated normal vectors on a sharp edge

Therefore, we use a second angle β, that determines the maximum angle between the normal vector of a candidate point and a reference vector for the segment. The reference vector $\vec{N}_{ref}$ is defined as the average of all normal vectors that belong to the current segment. After inserting a new point, the reference vector is updated. This angle criterion guarantees, that the angle between two arbitrary normal vectors in one segment is smaller or equal than 2β, fig. 4.2.

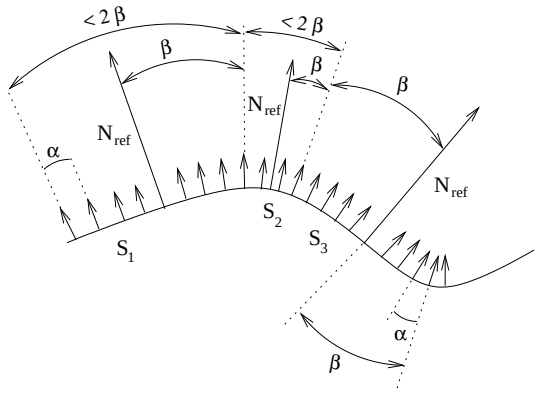


Figure 4.2: Segmentation using two angles

4.2 Cleaning up the segmentation

The segmentation method described in the preceding section has the following undesirable property: besides the larger segments (of a size mainly controlled by β) that provide a reasonable segmentation of the data set, a huge number of small segments

(number of points < 20) are produced that are certainly unwanted. One reason for the appearance of these segmentation artifacts is illustrated in fig. 4.3. There, one would expect the small segment S_2 to be part of the larger segment S_1. However, the segments are separated based on the β-angle criterion, because the reference normal N_{ref} of S_1 is strongly affected by the normals on the right side of S_1. Obviously, these normals have been considered in the segmentation procedure before the normals now belonging to S_2 have been checked.

In order to reduce the number of small segments we implemented three procedures to clean up the segmentation. The first step of these procedures joins a small segments with a neighboring larger one if this process violates the β-criterion only by a prescribed tolerance ε. More precisely we proceed as follows:

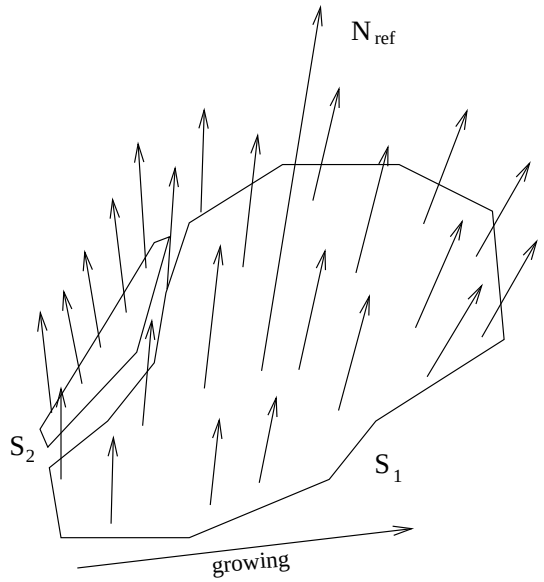


Figure 4.3: An example for segment creation

Algorithm: Cleaning up, pass 1

For all small segments S_f do:

- Consider the set $\mathcal{NP}(S_f)$ of all neighbors of all edge points of S_f. Store for each segment the number of points of $\mathcal{NP}(S_f)$ that it contains. We call this entry the neighborhood index.

- Examine all segments with a neighborhood index bigger than a specified threshold (e.g. 40%). Among these find the segment S with the smallest angle deviation η between its reference normal and reference normal of S_f. If $\eta < \beta + \varepsilon$ join S_f and S. Update the reference normal of S.

Remaining small segments are processed by a second procedure that intends to reduce the size of a small segment by repeatedly extracting its edge points.

Cleaning up, pass 2:

Let S_1 denote a small segment:

- Search for an edge point P of S_1 with a neighbor Q that belongs to a large segment S_2 with reference normal N_{ref}.
- If such a point can be found and the relations $\angle(N_P, N_Q) < \alpha \wedge \angle(N_P, N_Q) < \beta + \varepsilon$ hold, then add P to S_2 and update the list of edge points of S_1.
- Repeat until the list of edge points of S_1 is empty or all of its points are marked as not extractable.

Because the remaining small segments have not great importance for the surface fitting, in the third cleaning up procedure we collect all these segment to one segment which we call *remainder segment*, see fig. 4.5. Note, that the points of the remainder segment occur

a) close to sharp edges,

b) on the regions with sampling flaws,

c) on very noisy parts of regular segments.

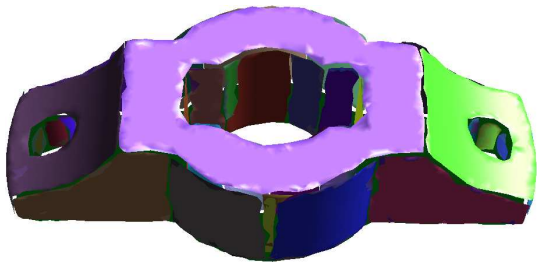


Figure 4.4: First order segmentation for artificially generated object consisting of 22,211 points. 41 segments result containing 21,112 points

4.2.1 Iterating segmentation and normals estimation

We already mentioned that the normal vectors close to the edges are estimated inaccurately because the neighborhoods contain points on both sides of the edge. This is especially the case if the data set results from a poor sampling that requires the use of large neighborhoods. These normal vectors often disturb the segmentation process and cause a large number of small segments. After the second order segmentation a postprocessing and surface recognition step is performed, which needs well estimated normal vectors for the first approximation of the type of the segment being examined.

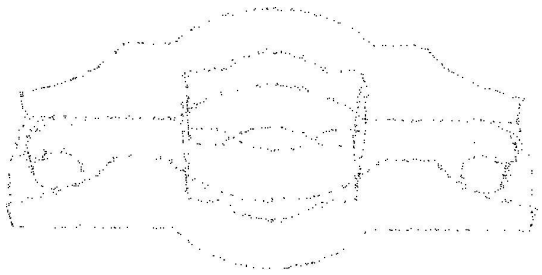


Figure 4.5: The remainder segment of the object on the figure 4.4

Therefore we implemented a procedure for normal vector recomputing, which searches for the points in the vicinity of sharp edges and improves the estimation of their normal vectors by modification of the point neighborhood.

Since the current segmentation provides rough information about the object's surface, we can approximately detect edges or regions with high curvature. Thus, the neighborhood of a point P in a regular segment can be temporarily adjusted, so that $\mathcal{N}(P)$ does not contain neighbors beyond sharp edges.

For every segment a list of its segment neighbors is created and the normal recomputing procedure proceeds as follows:

- Take a point P_i of a segment S_j.
- Mark all segment neighbors S_m of S_j, with the property $\angle(N_{ref}(S_m), N_{ref}(S_j)) < \delta$, where $N_{ref}(S_m)$ and $N_{ref}(S_j)$ are the reference vectors of the segments S_m and S_j resp. and δ is a threshold.
- Check if all neighbors of P_i belong to the marked segments. If not, remove them from $\mathcal{N}(P_i)$.
- If the neighborhood was not changed, continue with a next point. Otherwise extend $\mathcal{N}(P_i)$ to a full neighborhood as follows:
  - Take the neighbors of the points in $\mathcal{N}(P_i)$, which belong to marked segments and have not yet been included into $\mathcal{N}(P_i)$.
- Estimate the normal vector of P from the modified neighborhood using one of the described estimation methods and thereafter discard the modified neighborhood.

The procedure recomputes only normal vectors of regular segments. Normal vectors of the points in the remainder segment have to be recomputed in another way, because they are located mainly on the edges (or badly sampled areas) and it is impossible to determine the new modified neighborhood. In our implementation we try to smooth them by increasing the neighborhood.

The whole process (segmentation with subsequent recomputing) can be repeated, but as the tests have shown, more than two repetitions gives only very small improvements.

The recomputing works very fast (the worst case complexity is $O(n)$, but in general only normal vectors of edge points are recomputed) and for an object with many sharp edges it provides a good improvement of the normal vector estimation, see table 4.1 for an example of recomputing improvements for a box (the values in the table cells correspond to average resp. maximum angle between the estimated and the exact normal vector). If the transition between two segments is smooth, the normal vectors of the boundary points of these segments were well estimated with the initial estimation procedure and it is undesirable to recompute these normal vectors. It is the role of the angle δ to avoid the modification of the neighborhoods on this situation.

#Points	Initial Estimation	1. Recomputing	2. Recomputing
Noiseless data set: 10 neighbors			
10*k*	3.093 / 85.6	1.638 / 89.9	1.373 / 89.9
100*k*	1.006 / 87.8	0.5557 / 89.9	0.457 / 89.9
Noise: 100%, 0.1% of a bounding box's diagonal: 15 neighbors			
10*k*	4.219 / 82.53	2.299 / 89.9	1.817 / 89.9
100*k*	2.714 / 82.5	2.158 / 89.9	2.037 / 90
Noise: 100%, 0.1% of a bounding box's diagonal: 20 neighbors			
10*k*	4.72 / 69.03	2.526 / 89.9	1.943 / 89.9
100*k*	2.382 / 71.9	1.746 / 89.9	1.599 / 89.9
	BFP	1. Recomputing	2. Recomputing
Noise: 100%, 0.3% of a bounding box's diagonal: 15 neighbors			
10*k*	4.901 / 86.22	3.36 / 89.9	3.136 / 89.9
100*k*	4.891 / 88.8	4.488 / 89.9	4.436 / 89.9
Noise: 100%, 0.3% of a bounding box's diagonal: 20 neighbors			
10*k*	5.127 / 76.6	3.119 / 89.9	2.924 / 89.9
100*k*	4.056 / 88.7	3.542 / 89.9	3.477 / 90
Noise: 100%, 0.3% of a bounding box's diagonal: 25 neighbors			
10*k*	5.423 / 72	3.431 / 89.4	3.242 / 89.4
100*K*	3.628 / 81.5	3.082 / 90	3.06 / 90

Table 4.1: Normal recomputing for a box

Figure 4.6 shows the effect of normal vector recomputing for an object with a hole and many sharp edges.

Table 4.2 provides significant data of three passes of normal vectors computation and segmentation for a real object with 90,974 points. For each step the number of segments and the time performance in seconds is given (tested under Linux on a Athlon 1200 MHZ system with 512 MB DDR-RAM).

The tests have shown that many objects can be reliably segmented with a fixed choice of α and β. We use $\alpha = 8°, \beta = 20°$ as default values.

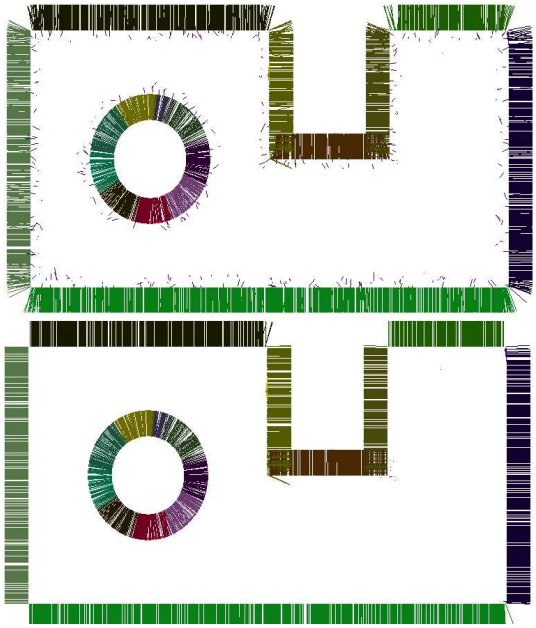


Figure 4.6: First step normal estimation (upper image) and the normal vectors after first recomputing (lower image)

4.3 Second order Segmentation

In the previous section we presented a reliable method for detection of sharp (i.e. tangent discontinuous) or nearly sharp edges (i.e. tangent continuous ones with a very high variation of normals). Our first order segmentation involves the following completeness property: any sharp or nearly sharp edge belongs to two different segments. This property assures that in the process of the normal recomputing the neighborhoods are correctly truncated to one side of a critical edge.

The completeness property is obtained by introducing the second control angle β, that refers to the deviation of a point normal to the reference vector of a segment. However, it is this angle that leads to an unwanted tessellation of smooth surfaces (e.g. surfaces of revolution) into several strips (see fig. 4.7). This disadvantage could be overcome by computing a final segmentation (with highly accurate normals due to the previous steps) that involves only the angle α. However, it is advisable to perform this additional pass of the segmentation based on second order information, since this allows to include tangent continuous but curvature discontinuous edges into the model.

In this section we will describe our second order segmentation strategy that is based on the notion of *principle curvatures* and *region growing* algorithm.

Real object, 90,974 points		
Process		Time
Searching (12 neighbors)		5.23
Normal vector estimation		3.51
Normal vector orientation		0.66
I. Segmentation	Found 3,428 seg.	0.69
	I. Clean up: -1,562 seg.	
	II. Clean up: -438 seg.	
	76 regular seg.:, 88,794 p.	
	Remainder seg.: 2,180 p.	
II. Segmentation	Found 1,837 seg.	0.61
	I. Clean up: -677 seg.	
	II. Clean up: -75 seg.	
	77 regular seg.: 89,597 p.	
	Remainder seg.: 1,377 p.	
III. Segmentation	Found 1,216 seg.	0.59
	I. Clean up: -144 seg.	
	II. Clean up: -22 seg.	
	79 regular seg.: 89,653 p.	
	Remainder seg.: 1,321 p.	
Whole processing time		14.57

Table 4.2: Automatic segmentation procedure with recomputing of the normal vectors after each segmentation

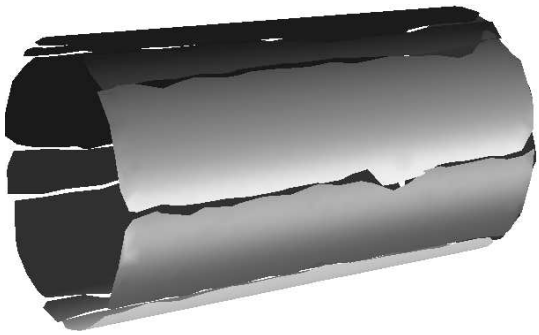


Figure 4.7: Subdivision of a surface of revolution into strips during first order segmentation

4.3.1 Processing principal curvatures

Consider two neighboring points P_1, P_2 on a discretized piecewise C^2 surface $\mathcal{S}$ and assume that these points are not separated by a sharp edge. For each point P_i there exist two directions $\mathbf{v}_{\min}(P_i), \mathbf{v}_{\max}(P_i)$ of minimum and maximum normal curvature $\kappa_{\min}(P_i), \kappa_{\max}(P_i)$ (the principal curvatures). If the second fundamental form of $\mathcal{S}$ (that

comprises all curvature information of the surface) changes continuously from P_1 to P_2, the closeness of these points suggests to assume that the differences $|\kappa_{\min}(P_1) - \kappa_{\min}(P_2)|$ and $|\kappa_{\max}(P_1) - \kappa_{\max}(P_2)|$ will be small.

We will therefore prescribe two thresholds $\Delta\kappa_{\min}$ and $\Delta\kappa_{\max}$ that correspond to the maximum deviation of the principal curvatures for two neighboring points to be accepted as points of one curvature continuous segment.

In section 3.5 the equations for computing the principal curvatures have been given. Since curvatures are more sensitive to noise in the data than normals, it is necessary to choose larger neighborhoods for curvature estimation than for normal estimation. Note, that these neighborhoods are correctly truncated to lie completely on one side of a sharp edge. However, the first order segmentation cannot locate tangent continuous but curvature discontinuous edges (e.g. between a cylinder and a sphere). Therefore a neighborhood for curvature estimation can cross such an edge. This will lead to wrongly estimated curvatures that vary smoothly across the edge. Figure 4.8 illustrates this problem: in the dotted area the neighborhood of a point P contains points of both surfaces and therefore the minimum curvature smoothly varies from 0 (cylinder) to $1/R$ (sphere).

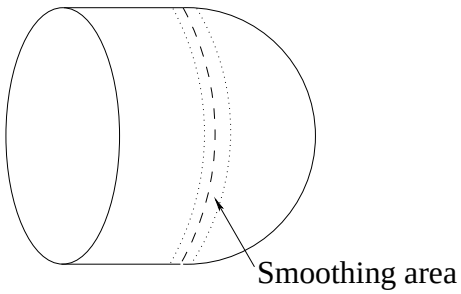


Figure 4.8: Curvatures smoothing problem between smooth transition between a cylinder and a sphere

In analogy to our approach in the first order case we introduce two additional parameters into the segmentation procedure. First we define a reference minimum and reference maximum curvature of a segment as the average of any value $\kappa_{\min}(P)$ and $\kappa_{\max}(P)$ within that segment: $\kappa^{ref}_{\min} = \frac{1}{n_s}\sum_{i=1}^{n_s}\kappa_{\min}(P_i)$, $\kappa^{ref}_{\max} = \frac{1}{n_s}\sum_{i=1}^{n_s}\kappa_{\max}(P_i)$. Then we only accept a point P_i to be included into a segment, if the differences $|\kappa^{ref}_{\min} - \kappa_{\min}(P_i)|$ and $|\kappa^{ref}_{\max} - \kappa_{\max}(P_i)|$ stay smaller than the prescribed thresholds $\Delta\kappa^{ref}_{\min}, \Delta\kappa^{ref}_{\max}$ respectively.

The segmentation proceeds by initializing a segment by a single point and recursively checking the neighbors of any point in the segment to fulfill the prescribed conditions. If a neighboring point is found that satisfies the four segmentation criteria described above, the point is included in the segment. If no such point can be found the segment is considered to be complete and a new segment is initialized.

The starting point for any new segment is taken from a sorted point list. This list contains all data points sorted in ascending lexicographical order of the absolute values of both curvatures $\kappa_{\min}$ and $\kappa_{\max}$, i.e. $\text{Index}(P) < \text{Index}(Q)$ iff $|\kappa_{\min}(P)| < |\kappa_{\min}(Q)|$ or

$|\kappa_{\min}(P)| = |\kappa_{\min}(Q)|$ and $|\kappa_{\max}(P)| < |\kappa_{\max}(Q)|$. Within this list any point that already belongs to a segment is flagged. The first unflagged entry is chosen as the starting point for a new segment.

This choice assures that the segmentation will proceed from simple to more complex geometries, a fact that is advantageous in many situations.

Consider two adjacent tangent continuous surfaces of different curvature (e.g. a cylinder and a plane). Our choice guarantees that the segmentation will be initialized in the interior of the plane (small curvatures) while an arbitrary choice may have initialized a segment at the boundary between the surfaces. The first choice would result in a mainly planar segment that slightly enters into the cylinders. The second choice however would create a small segment that would be resolved in the following post-processing steps.

The segmentation terminates when all entries in the sorted point list are flagged.

In contrast to the first order segmentation that provides very good results on a vast class of objects for a fixed choice of values α, β (see section 4.1), the quality of the second order segmentation highly depends on the choice of the four thresholds $\Delta\kappa_{\min}, \Delta\kappa_{\max}, \Delta\kappa_{\min}^{ref}, \Delta\kappa_{\max}^{ref}$. Up to date we do not have a fully reliable procedure to determine these thresholds automatically for all object types. Therefore the parameters provided by our semi-automatic system has to be supervised (if needed) by a human.

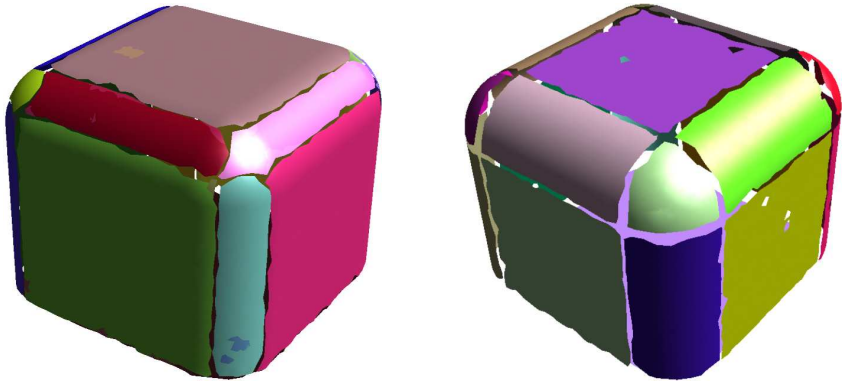
To determine these thresholds we have tested several methods. The most successful one is based on the knowledge provided by the first order segmentation, i.e. we compute the curvature mean ($\overline{\kappa}_{\min}$ and $\overline{\kappa}_{\max}$) of the points in a regular segment S_i (created during the first order segmentation): $\overline{\kappa}_{\min} = \frac{1}{n_i}\sum_{j=1}^{n_i}\kappa(P_j)$ ($\overline{\kappa}_{\max}$ analogously) and thereafter the average curvature of all curvature means of regular segments: $\kappa_{\min}^{\text{ave}} = \frac{1}{n_s}\sum_{i=1}^{n_s}\overline{\kappa}_{\min}^{i}$ (analogously $\kappa_{\max}^{\text{ave}}$). Then the thresholds $\Delta\kappa_{\min} = t_1 * \kappa_{\min}^{\text{ave}}$, $\Delta\kappa_{\max} = t_1 * \kappa_{\max}^{\text{ave}}$ and $\Delta\kappa_{\min}^{ref} = t_2 * \kappa_{\min}^{\text{ave}}$, $\Delta\kappa_{\min}^{ref} = t_2 * \kappa_{\min}^{\text{ave}}$. The coefficients t_1 and t_2 have been estimated empirically and depend on rough noise estimation provided by the user, see section 3.3. If the user finds by inspection that geometrical elements as planes, cylinders or spheres are subdivided into many segments $\Delta\kappa_{\min}$ and $\Delta\kappa_{\max}$ are repeatedly increased until a satisfactory result is obtained. Analogously if two smooth surfaces[1] are joined together, $\Delta\kappa_{\min}$ and $\Delta\kappa_{\max}$ have to be decreased, until a correct disconnection is achieved.

For simple objects with a few distinct surfaces (up to 30) the parameters provided by our semi-automatic system do not have to be modified, the segmentation behaves correctly, see figures 4.9, 4.10 and 4.11 . Only big (noisy) objects with many surfaces with different sizes require user interaction, if necessary multiple passes of the combination segmentation-postprocessing.

All the three figures 4.9, 4.10 and 4.11 are artificially created point sets, that we contaminated with random noise before processing the data set.

The data set on the figure 4.9 was contaminated slightly with noise (0.05% of the bounding box's diagonal). This data set is simple, but it consists prevailingly of smooth (C^1) transitions. The first order segmentation **a)** was not able to find any simple surface

[1] Note, that the surfaces with C^0 transitions are always separated satisfactory, because the principal curvatures of the points on sharp edges are estimated fully wrong - in general they are 10–1000 times bigger than the curvatures of the adjacent points on regular surfaces.



a) b)

Figure 4.9: First order **a)** and second order **b)** segmentation of a curved box

a) b)

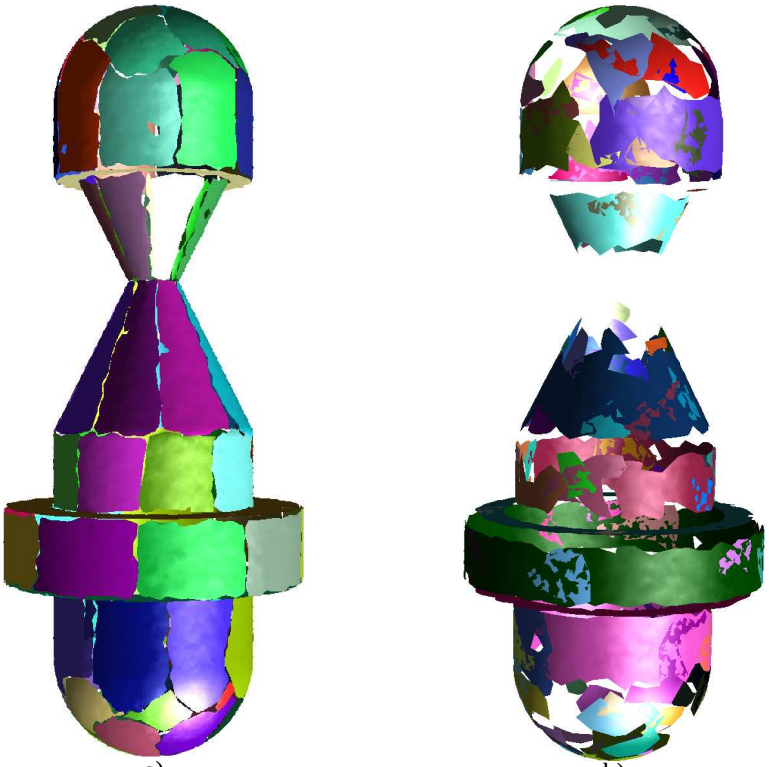
Figure 4.10: First order **a)** and second order **b)** segmentation of a mechanic part

(even the 'planar' segments contain some points which should belong to the adjacent spheres or cylinders).

For the data set on the fig. 4.10 we increased the amount of noise added to every point to 0.1% of the bounding box's diagonal length. The gaps between adjacent segments became bigger, but all object's surfaces were segmented correctly.

The object on the fig. 4.11 was contaminated with strong noise: 0.3% of the diameter of the biggest cylinder. The noise caused creation of a huge number of segments, but on the other side this initial second order segmentation was sufficient for the subsequent step to recognize and enlarge all the object's surfaces properly.

Note that both segmentations of all three objects result from default segmentation parameters provided by our semi-automatic system. The user only specified in the beginning the estimation of the noise in the data (in the scale from 0–100), 25 for the data set in the fig. 4.9, 50 for the data set in the fig. 4.10 and 75 for the data set in the fig. 4.11. No further user interaction was necessary.



a) b)

Figure 4.11: First order **a)** and second order **b)** segmentation of a rotational part

Chapter 5

Postprocessing and Surface Fitting

The presented second order segmentation is generally acceptable but it contains flaws due to the following problems:

- If the data is significantly contaminated with noise, the curvature estimation is not very precise. In this case it may happen that a surface of a certain type (e.g. cylindrical) is segmented into several pieces; usually a larger segment of the correct type and some small segments of the same or different type, see fig. 4.11 b).
- Even if the data is free of noise, the curvature estimation is problematic in the vicinity of tangent continuous but curvature discontinuous edges as explained in the section 4.3.1. Therefore, we cannot expect a precise segmentation in those regions. See the gaps between the segmented surfaces in the fig. 4.9.
- In any case, a cone will be divided into several slices due to the threshold on the maximum reference curvature. Here, it is certainly desirable to merge these slices into one segment, see fig. 5.1.

In this chapter we will present a postprocessing strategy to overcome these segmentation flaws. Our approach is based on the recognition of the following simple algebraic surfaces: planes, cylinders, cones and spheres.

Before we introduce the whole postprocessing procedure, in the next sections we will describe how we obtain the algebraic equations (or "algebraic parameterizations") of the four mentioned simple surfaces.

5.1 Plane fitting

The plane fitting is based on the well known "principal component analysis" [28] (see section 2.4). The plane is defined by the centre of mass of all segment points and

a) b)

Figure 5.1: Second order segmentation of a cone **a)** and the cone after postprocessing **b)**.

the normal vector (the eigenvector of the covariance matrix belonging to the smallest eigenvalue).

5.2 Sphere fitting

Our routine for a spherical fit is based on the approach proposed by Lukács [30]. In order to determine the algebraic equation of the sphere, the following distance function is minimized:

$$\begin{aligned} f(d) &= \sum_{i=1}^{n_s} d^2(s,\mathbf{P}_i) \\ d(s,\mathbf{P}_i) &= |\mathbf{P}_i - (\rho + R)N| - R = \\ &= \sqrt{\langle \mathbf{P}_i, \mathbf{P}_i \rangle - 2(\rho + R)\langle \mathbf{P}_i, N \rangle + (\rho + R)^2} - R \end{aligned}$$

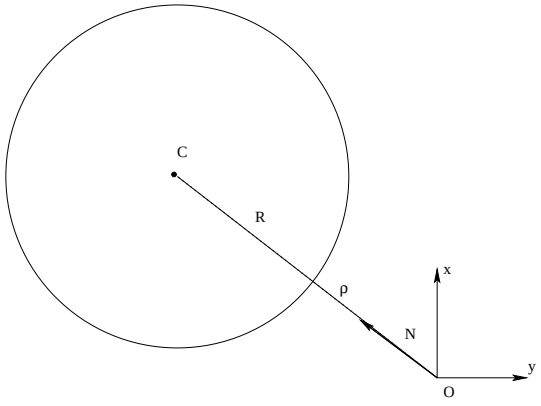


Figure 5.2: Parameterization of a sphere

where R is the radius of the sphere, $\rho\mathbf{N}$ is the closest point on the sphere to the origin ($\|\mathbf{N}\| = 1$), $(\rho + R)N$ is the centre of the sphere and $\mathbf{P}_i$ are the points, see fig. 5.2.

Lukács proposed minimization of a modified distance function $\tilde{d}(s,\mathbf{P}_i)$, where the square root of the distance does not occur. He proved that if $\tilde{d}(s,\mathbf{P}_i)$ vanishes, then $d(s,\mathbf{P}_i) \to 0$. That is of course true, but we observed that if $d(s,\mathbf{P}_i) > 0$, then $\tilde{d}(s,\mathbf{P}_i) \gg 0$ and the minimization procedure tends to perform more iterations to find the same minimum (he further cumulates the error by computing the gradient and the Hessian from the modified distance function $\tilde{d}(s,\mathbf{P}_i)$). It does not change the complexity of the algorithm, but in every iteration the function value, gradient and Hessian of n_s points are computed. Therefore the minimization of $\tilde{d}(s,\mathbf{P}_i)$ is more time consuming than the minimization of $d(s,\mathbf{P}_i)$.

The next reason is that we use the distance function for accepting / rejecting the segment as spherical, cylindrical or conical segment, if the average distance of the points to the surface (i.e. $D_A = \frac{1}{n_s}\sum_{i=1}^{n_s} |d(s,\mathbf{P}_i)|$) is less / greater than a prescribed threshold. We observed, that the usage of the exact distance function is more stable for classification of noisy segments (objects).

A Newton-Raphson method, which requires the gradient and Hessian, was used for the minimization. The advantage is very fast convergence if we provide a decent estimation of the solution vector.

As we show in the next, the evaluation time can be optimized, as only 3 terms for the gradient and 6 terms for Hessian need to be computed.

Let

$$\begin{aligned}
SQRT &= \sqrt{\langle \mathbf{P}_i,\mathbf{P}_i\rangle - 2(\rho+R)\langle \mathbf{P}_i,N\rangle + (\rho+R)^2} \\
N &= (\cos\varphi\sin\vartheta, \sin\varphi\sin\vartheta, \cos\vartheta) \\
N^{\varphi} &= (-\sin\varphi\sin\vartheta, \cos\varphi\sin\vartheta, 0) \\
N^{\vartheta} &= (\cos\varphi\cos\vartheta, \sin\varphi\cos\vartheta, -\sin\vartheta) \\
N^{\varphi\varphi} &= (-\cos\varphi\sin\vartheta, -\sin\varphi\sin\vartheta, 0) \\
N^{\vartheta\vartheta} &= -N \\
N^{\varphi\vartheta} &= (-\sin\varphi\cos\vartheta, \cos\varphi\cos\vartheta, 0) \\
N^{\vartheta\varphi} &= N^{\varphi\vartheta}
\end{aligned}$$

Note that all these variables (except for $SQRT$) need to be computed once during one iteration, as they are independent on $\mathbf{P}_i$. If the point $\mathbf{P}_i$ does not coincide with the sphere centre, $SQRT \neq 0$ and all partial derivations are defined.

The gradient $\nabla d^2(s,\mathbf{P}_i)$ can be easily computed from

$$\begin{aligned}
\frac{\partial d(s,\mathbf{P}_i)}{\partial\rho} &= \frac{\rho+R-\langle \mathbf{P}_i,N\rangle}{SQRT} \\
\frac{\partial d(s,\mathbf{P}_i)}{\partial\varphi} &= -\frac{(\rho+R)\langle \mathbf{P}_i,N^{\varphi}\rangle}{SQRT} \\
\frac{\partial d(s,\mathbf{P}_i)}{\partial\vartheta} &= -\frac{(\rho+R)\langle \mathbf{P}_i,N^{\vartheta}\rangle}{SQRT}
\end{aligned}$$

$$\frac{\partial d(s,\mathbf{P}_i)}{\partial R} = \frac{\partial d(s,\mathbf{P}_i)}{\partial \rho} - 1$$

as

$$\begin{aligned}\nabla f(d) &= \sum_{i=1}^{n_s}\left[\frac{\partial d^2(s,\mathbf{P}_i)}{\partial \rho},\frac{\partial d^2(s,\mathbf{P}_i)}{\partial \varphi},\frac{\partial d^2(s,\mathbf{P}_i)}{\partial \vartheta},\frac{\partial d^2(s,\mathbf{P}_i)}{\partial R}\right]\\ &= 2\sum_{i=1}^{n_s} d(s,\mathbf{P}_i)\left[\frac{\partial d(s,\mathbf{P}_i)}{\partial \rho},\frac{\partial d(s,\mathbf{P}_i)}{\partial \varphi},\frac{\partial d(s,\mathbf{P}_i)}{\partial \vartheta},\frac{\partial d(s,\mathbf{P}_i)}{\partial R}\right]\end{aligned}$$

The Hessian matrix can be computed from

$$\begin{aligned}\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho^2} &= SQRT^{-1} - \frac{(\rho+R-\langle \mathbf{P}_i,N\rangle)^2}{SQRT^3}\\ \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho \partial \varphi} &= -\frac{\langle \mathbf{P}_i,N^{\varphi}\rangle}{SQRT} + \frac{(\rho+R-\langle \mathbf{P}_i,N\rangle)(\rho+R)\langle \mathbf{P}_i,N^{\varphi}\rangle}{SQRT^3}\\ \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho \partial \vartheta} &= -\frac{\langle \mathbf{P}_i,N^{\vartheta}\rangle}{SQRT} + \frac{(\rho+R-\langle \mathbf{P}_i,N\rangle)(\rho+R)\langle \mathbf{P}_i,N^{\vartheta}\rangle}{SQRT^3}\\ \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho \partial R} &= \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho^2}\\ \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \varphi^2} &= -\frac{(\rho+R)\langle \mathbf{P}_i,N^{\varphi\varphi}\rangle}{SQRT} - \frac{(\rho+R)^2\langle \mathbf{P}_i,N^{\varphi}\rangle^2}{SQRT^3}\\ \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \varphi \partial \vartheta} &= -\frac{(\rho+R)\langle \mathbf{P}_i,N^{\varphi\vartheta}\rangle}{SQRT} - \frac{(\rho+R)^2\langle \mathbf{P}_i,N^{\varphi}\rangle\langle \mathbf{P}_i,N^{\vartheta}\rangle}{SQRT^3}\\ \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \varphi \partial R} &= \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho \partial \varphi}\\ \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \vartheta^2} &= -\frac{(\rho+R)\langle \mathbf{P}_i,N^{\vartheta\vartheta}\rangle}{SQRT} - \frac{(\rho+R)^2\langle \mathbf{P}_i,N^{\vartheta}\rangle^2}{SQRT^3}\\ \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \vartheta \partial R} &= \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho \partial \vartheta}\\ \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial R^2} &= \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho^2}\end{aligned}$$

Note that

$$\frac{\partial^2(s,\mathbf{P}_i)}{\partial x_j \partial x_k} = \frac{\partial^2(s,\mathbf{P}_i)}{\partial x_k \partial x_j}.$$

as

$$H = \left[\frac{\partial^2 d^2(s,\mathbf{P}_i)}{\partial x_j \partial x_k}\right]_{j,k=1,\ldots,4}$$

where

$$\frac{\partial^2 d^2(s,\mathbf{P}_i)}{\partial x_j \partial x_k} = 2\left(\frac{\partial d(s,\mathbf{P}_i)}{\partial x_j}\frac{\partial d(s,\mathbf{P}_i)}{\partial x_k} + d(s,\mathbf{P}_i)\frac{\partial^2(s,\mathbf{P}_i)}{\partial x_j \partial x_k}\right)$$

5.2.1 Obtaining the sphere parameterization from the segmentation

In the previous chapters we described our estimation methods for various surface or segment information. For a segment S_i we store in our data structures all its points $P_j, j = 1,\ldots,n_s$ with their normal vectors $N(P_j)$ and principal curvatures $\kappa_{\min}(P_j), \kappa_{\max}(P_j)$, and segment's reference curvatures $\kappa^{\text{ref}}_{\min}(S_i), \kappa^{\text{ref}}_{\max}(S_i)$.

Using stored data we can compute the first estimation of the solution vector $s(\rho, R, \varphi, \vartheta)$ of the distance function $d(s,\mathbf{P}_i)$. In the beginning we determine the orientation of the normal vectors (pointing inside or outside the sphere) and set a coefficient $f = 1$ if they point inside the sphere or $f = -1$ if they point outside the sphere. We estimate the radius as

$$R = \frac{2}{\kappa^{\text{ref}}_{\min}(S_i) + \kappa^{\text{ref}}_{\max}(S_i)}$$

The sphere centre is computed as $C = \frac{1}{n_s}\sum_{j=1}^{n_s}(P_j + f\,R\,N(P_j))$. Further: $N(\varphi,\vartheta) = \frac{C}{\|C\|}$ and $\rho = \|C\| - R$.

In general, for spherical segments[1] this first estimation of $s(\rho, R, \varphi, \vartheta)$ is not far from the minimal one, so that 1–5 iterations are necessary to find the global minimum.

5.3 Cylinder fitting

For the cylinder fitting we again use the parameterization proposed by Lukács [30] with the same modification as mentioned in the previous section 5.2.

The parameterization of the cylinder is illustrated on the fig. 5.3. We minimize the following exact distance function:

$$\begin{aligned} f(d) &= \sum_{i=1}^{n_s} d^2(s,\mathbf{P}_i) \\ d(s,\mathbf{P}_i) &= |(\mathbf{P}_i - (\rho+R)N)\times A| - R \\ &= \sqrt{\langle \mathbf{P}_i,\mathbf{P}_i\rangle - 2(\rho+R)\langle \mathbf{P}_i, N\rangle + (\rho+R)^2 - \langle \mathbf{P}_i, A\rangle^2} - R \end{aligned}$$

where R is the radius of the cylinder, A is the cylinder axis vector, N is a vector perpendicular to A, ρN is the nearest point on the cylinder surface from the origin O and $(\rho+R)N$ is a point on the cylinder axis.

As for the sphere the Newton-Raphson minimization method was used.

[1] For noise data sets it often happens that we run the minimization procedure for non-spherical segments. After minimization the distance test is performed (average distance of the points to the surface) which rejects such a segment as a non-spherical.

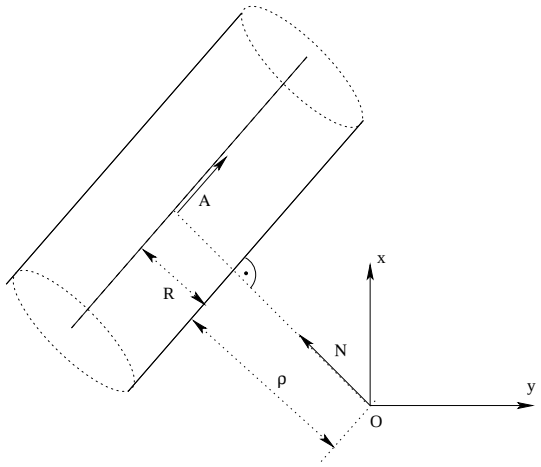


Figure 5.3: Parameterization of a cylinder

Let $N, N^{\varphi}, N^{\vartheta}, N^{\varphi\varphi}, N^{\vartheta\vartheta}, N^{\varphi\vartheta}, N^{\vartheta\varphi}$ be as defined in the section 5.2, and further

$$\begin{aligned}
SQRT &= \sqrt{\langle \mathbf{P}_i, \mathbf{P}_i \rangle - 2(\rho + R)\langle \mathbf{P}_i, N \rangle + (\rho + R)^2 - \langle \mathbf{P}_i, A \rangle^2} \\
N^{\vartheta\varphi\varphi} &= (-\cos\varphi\cos\vartheta, -\sin\varphi\cos\vartheta, 0) \\
\overline{N^{\varphi}} &= (-\sin\varphi, \cos\varphi, 0) \\
\overline{N^{\varphi\varphi}} &= (-\cos\varphi, -\sin\varphi, 0) \\
A &= N^{\vartheta}\cos\alpha + \overline{N^{\varphi}}\sin\alpha \\
A^{\alpha} &= -N^{\vartheta}\sin\alpha + \overline{N^{\varphi}}\cos\alpha \\
A^{\varphi} &= N^{\vartheta\varphi}\cos\alpha + \overline{N^{\varphi\varphi}}\sin\alpha \\
A^{\vartheta} &= -N\cos\alpha \\
A^{\varphi\varphi} &= N^{\vartheta\varphi\varphi}\cos\alpha - \overline{N^{\varphi}}\sin\alpha \\
A^{\varphi\vartheta} &= -N^{\varphi}\cos\alpha \\
A^{\vartheta\vartheta} &= -N^{\vartheta}\cos\alpha \\
A^{\varphi\alpha} &= -N^{\vartheta\varphi}\sin\alpha + \overline{N^{\varphi\varphi}}\cos\alpha \\
A^{\vartheta\alpha} &= N\sin\alpha
\end{aligned}$$

Analogously to the sphere fitting, all these variables (except for *SQRT*) have to be computed only once during one iteration of the minimization procedure, as they do not depend on $\mathbf{P}_i$. As long as the point $\mathbf{P}_i$ does not lie on the cylinder axis, $SQRT \neq 0$ and all partial derivations are defined.

The first partial derivations for the gradient are:

$$\frac{\partial d(s, \mathbf{P}_i)}{\partial \rho} = \frac{\rho + R - \langle \mathbf{P}_i, N \rangle}{SQRT}$$

$$
\begin{aligned}
\frac{\partial d(s,\mathbf{P}_i)}{\partial \varphi} &= -\frac{(\rho+R)\langle \mathbf{P}_i, N^{\varphi}\rangle + \langle A,\mathbf{P}_i\rangle \langle A^{\varphi},\mathbf{P}_i\rangle}{SQRT} \\
\frac{\partial d(s,\mathbf{P}_i)}{\partial \vartheta} &= -\frac{(\rho+R)\langle \mathbf{P}_i, N^{\vartheta}\rangle + \langle A,\mathbf{P}_i\rangle \langle A^{\vartheta},\mathbf{P}_i\rangle}{SQRT} \\
\frac{\partial d(s,\mathbf{P}_i)}{\partial \alpha} &= -\frac{\langle A,\mathbf{P}_i\rangle \langle A^{\alpha},\mathbf{P}_i\rangle}{SQRT} \\
\frac{\partial d(s,\mathbf{P}_i)}{\partial R} &= \frac{\partial d(s,\mathbf{P}_i)}{\partial \rho} - 1
\end{aligned}
$$

The second partial derivation are necessary for computing of the Hessian:

$$
\begin{aligned}
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho^2} &= \frac{1}{SQRT} - \frac{(\rho+R-Scal\mathbf{P}_iN)^2}{SQRT^3} \\
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho \partial \varphi} &= \frac{(\rho+R-\langle \mathbf{P}_i,N\rangle)((\rho+R)\langle \mathbf{P}_i,N^{\varphi}\rangle + \langle A,\mathbf{P}_i\rangle\langle A^{\varphi},\mathbf{P}_i\rangle)}{SQRT^3} - \frac{\langle \mathbf{P}_i,N^{\varphi}\rangle}{SQRT} \\
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho \partial \vartheta} &= \frac{(\rho+R-\langle \mathbf{P}_i,N\rangle)((\rho+R)\langle \mathbf{P}_i,N^{\vartheta}\rangle + \langle A,\mathbf{P}_i\rangle\langle A^{\varphi},\mathbf{P}_i\rangle)}{SQRT^3} - \frac{\langle \mathbf{P}_i,N^{\vartheta}\rangle}{SQRT} \\
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho \partial \alpha} &= \frac{(\rho+R-\langle \mathbf{P}_i,N\rangle)\langle A,\mathbf{P}_i\rangle\langle A^{\alpha},\mathbf{P}_i\rangle}{SQRT^3} \\
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho \partial R} &= \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho^2}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \varphi^2} &= -\frac{(\rho+R)\langle \mathbf{P}_i,N^{\varphi\varphi}\rangle + \langle A^{\varphi},\mathbf{P}_i\rangle^2 + \langle A,\mathbf{P}_i\rangle\langle A^{\varphi\varphi},\mathbf{P}_i\rangle}{SQRT} \\
&\quad -\frac{((\rho+R)\langle \mathbf{P}_i,N^{\varphi}\rangle + \langle A,\mathbf{P}_i\rangle\langle A^{\varphi},\mathbf{P}_i\rangle)^2}{SQRT^3} \\
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \varphi \partial \vartheta} &= -\frac{(\rho+R)\langle \mathbf{P}_i,N^{\varphi\vartheta}\rangle + \langle A^{\vartheta},\mathbf{P}_i\rangle\langle A^{\varphi},\mathbf{P}_i\rangle + \langle A,\mathbf{P}_i\rangle\langle A^{\varphi\vartheta},\mathbf{P}_i\rangle}{SQRT} \\
&\quad -\frac{((\rho+R)\langle \mathbf{P}_i,N^{\varphi}\rangle + \langle A,\mathbf{P}_i\rangle\langle A^{\varphi},\mathbf{P}_i\rangle)\left((\rho+R)\langle \mathbf{P}_i,N^{\vartheta}\rangle + \langle A,\mathbf{P}_i\rangle\langle A^{\vartheta},\mathbf{P}_i\rangle\right)}{SQRT^3} \\
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \varphi \partial \alpha} &= -\frac{\langle A^{\alpha},\mathbf{P}_i\rangle\langle A^{\varphi},\mathbf{P}_i\rangle + \langle A,\mathbf{P}_i\rangle\langle A^{\varphi\alpha},\mathbf{P}_i\rangle}{SQRT} \\
&\quad -\frac{(\rho+R)\langle \mathbf{P}_i,N^{\varphi}\rangle + \langle A,\mathbf{P}_i\rangle\langle A^{\varphi},\mathbf{P}_i\rangle)\langle A,\mathbf{P}_i\rangle\langle A^{\alpha},\mathbf{P}_i\rangle}{SQRT^3} \\
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \varphi \partial R} &= \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \rho \partial \varphi}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial \vartheta^2} &= -\frac{\langle A^{\vartheta},\mathbf{P}_i\rangle^2 + \langle A,\mathbf{P}_i\rangle\langle A^{\vartheta\vartheta},\mathbf{P}_i\rangle - (\rho+R)\langle \mathbf{P}_i,N\rangle}{SQRT} \\
&\quad -\frac{((\rho+R)\langle \mathbf{P}_i,N^{\vartheta}\rangle + \langle A,\mathbf{P}_i\rangle\langle A^{\vartheta},\mathbf{P}_i\rangle)^2}{SQRT^3}
\end{aligned}
$$

$$\begin{aligned}
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial\vartheta\partial\alpha} &= -\frac{\langle A^\alpha,\mathbf{P}_i\rangle\left\langle A^\vartheta,\mathbf{P}_i\right\rangle + \langle A,\mathbf{P}_i\rangle\left\langle A^{\vartheta\alpha},\mathbf{P}_i\right\rangle}{SQRT} \\
&\quad -\frac{((\rho+R)\left\langle\mathbf{P}_i,N^\vartheta\right\rangle + \langle A,\mathbf{P}_i\rangle\left\langle A^\vartheta,\mathbf{P}_i\right\rangle)\langle A,\mathbf{P}_i\rangle\langle A^\alpha,\mathbf{P}_i\rangle}{SQRT^3} \\
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial\vartheta\partial R} &= \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial\rho\partial\vartheta} \\
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial\alpha^2} &= \frac{\langle A,\mathbf{P}_i\rangle^2 - \langle A^\alpha,\mathbf{P}_i\rangle^2}{SQRT} - \frac{\langle A,\mathbf{P}_i\rangle^2\langle A^\alpha,\mathbf{P}_i\rangle^2}{SQRT^3} \\
\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial R^2} &= \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial\rho^2}
\end{aligned}$$

Because of the symmetry and the property that $\frac{\partial^2 d(s,\mathbf{P}_i)}{\partial R\partial x_j} = \frac{\partial^2 d(s,\mathbf{P}_i)}{\partial\rho\partial x_j}$ only 10 items (of 25) of the Hessian matrix have to be computed.

5.3.1 Obtaining the cylinder parameterization from the segmentation

In the beginning the orientation of the normal vectors (inside or outside the cylinder) has to be determined. Let $f=-1$, if they point outside the cylinder, $f=1$ vice versa. The cylinder radius can be estimated using the segment's maximum reference curvature: $R = 1/\kappa_{\max}^{\text{ref}}(S_j)$ (for the segment parameters, see the section 5.2.1). A point on the cylinder axis can be computed as $P_A = \frac{1}{n_s}\sum_{j=1}^{n_s}(P_j + f.R.N(P_j))$.

We assign the cylinder axis vector A as the normal vector of the best fit plane of the end points of the normal vectors projected on the Gaussian sphere.

Now

$$\begin{aligned}
N(\varphi,\vartheta) &= \frac{P_A - \langle A,P_A\rangle}{\|P_A - \langle A,P_A\rangle\|} \\
\rho &= \|P_A - \langle A,P_A\rangle\| - R
\end{aligned}$$

and $\alpha = \angle(A,N^\vartheta)$.

Thus, we obtained the first estimation of the solution vector $s(\rho,\varphi,\vartheta,\alpha,R)$ of the distance function $d(s,\mathbf{P}_i)$ and the minimization procedure can be started.

5.4 Cone fitting

To get an equation of the cone we again use the parameterization by Lukács [30]. Because the equation of the cone is much more complicated as for the sphere or cylinder and the cone is more noise sensitive as the previous surfaces, it is natural to use the exact distance function. The Lukács' approximation $\tilde{d}(s,\mathbf{P}_i)$ of the distance function evinced less stability against noisy segments.

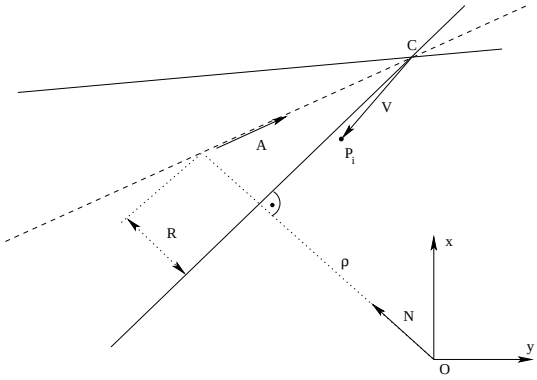


Figure 5.4: Parameterization of a cylinder

The parameterization of the cone is illustrated in the fig. 5.4. C is the cone apex, ρN is the nearest point from origin to the cone mantle, $(\rho+R)N$ is a point on the cone axis, A is the cone axis vector, pointing from the cone interior to the cone apex, V is the vector from the cone apex to a point $\mathbf{P}_i$.

We minimize the following distance function:

$$\begin{aligned} f(d) &= \sum_{i=1}^{n_s} d^2(s,\mathbf{P}_i) \\ d(s,\mathbf{P}_i) &= |(\mathbf{P}_i - C)\times A||N\times A| - |\langle \mathbf{P}_i - C, A\rangle \langle N, A\rangle| \\ &= \sqrt{(\langle V,V\rangle - \langle V,A\rangle^2)(1-\langle N,A\rangle^2)} - |\langle V,A\rangle \langle N,A\rangle| \end{aligned}$$

Let $N, N^\varphi, N^\vartheta$ be as defined in the section 5.2, further we set

$$\begin{aligned} SQRT &= \sqrt{(\langle V,V\rangle - \langle V,A\rangle^2)(1-\langle N,A\rangle^2)} \\ SIGN &= \mathrm{SGN}(\langle V,A\rangle \langle N,A\rangle) \\ A &= (\cos\sigma\sin\tau, \sin\sigma\sin\tau, \cos\tau) \\ A^\sigma &= (-\sin\sigma\sin\tau, \cos\sigma\sin\tau, 0) \\ A^\tau &= (\cos\sigma\cos\tau, \sin\sigma\cos\tau, -\sin\tau) \\ V &= \mathbf{P}_i - (\rho+R)N + \frac{R}{\langle N,A\rangle}A \\ V^\varphi &= -(\rho+R)N^\varphi - \frac{R\langle N^\varphi, A\rangle}{\langle N,A\rangle^2}A \\ V^\vartheta &= -(\rho+R)N^\vartheta - \frac{R\langle N^\vartheta, A\rangle}{\langle N,A\rangle^2}A \\ V^\sigma &= \frac{R}{\langle N,A\rangle}A^\sigma - \frac{R\langle N, A^\sigma\rangle}{\langle N,A\rangle^2}A \end{aligned}$$

$$
\begin{aligned}
V^{\tau} &= \frac{R}{\langle N,A\rangle}A^{\tau} - \frac{R\langle N,A^{\tau}\rangle}{\langle N,A\rangle^2}A \\
V^{\rho} &= -N \\
V^{R} &= -N + \frac{1}{\langle N,A\rangle}A
\end{aligned}
$$

Only for the case $\mathbf{P}_i = C$ is $SQRT = 0$. If this occurs, the point has to be excluded from the computation, because the partial derivations would not be defined. In all other cases the gradient is well defined. The function $\mathrm{SGN}(a)$ yields the sign of the argument a, i.e. -1 if a is negative, 1 if a is positive or 0 if $a = 0$.

Except for $SQRT$, $SIGN$ and V all the terms are independent on $\mathbf{P}_i$ and therefore they have to be computed only once during one iteration.

For the gradient we need the first partial derivations of the distance function:

$$
\begin{aligned}
\frac{\partial d(s,\mathbf{P}_i)}{\partial \varphi} &= \frac{(\langle V^{\varphi},V\rangle - \langle V,A\rangle\langle V^{\varphi},A\rangle)(1-\langle N,A\rangle^2) - |V\times A|^2\langle N,A\rangle\langle N^{\varphi},A\rangle}{SQRT} \\
&\quad -SIGN\left(\langle V^{\varphi},A\rangle\langle N,A\rangle + \langle V,A\rangle\langle N^{\varphi},A\rangle\right) \\
\frac{\partial d(s,\mathbf{P}_i)}{\partial \vartheta} &= \frac{(\langle V^{\vartheta},V\rangle - \langle V,A\rangle\langle V^{\vartheta},A\rangle)(1-\langle N,A\rangle^2) - |V\times A|^2\langle N,A\rangle\langle N^{\vartheta},A\rangle}{SQRT} \\
&\quad -SIGN\left(\left\langle V^{\vartheta},A\right\rangle\langle N,A\rangle + \langle V,A\rangle\left\langle N^{\vartheta},A\right\rangle\right) \\
\frac{\partial d(s,\mathbf{P}_i)}{\partial \sigma} &= \frac{(\langle V^{\sigma},V\rangle - \langle V,A\rangle(\langle V^{\sigma},A\rangle + \langle V,A^{\sigma}\rangle))(1-\langle N,A\rangle^2) - |V\times A|^2\langle N,A\rangle\langle N,A^{\sigma}\rangle}{SQRT} \\
&\quad -SIGN\left(\langle V^{\sigma},A\rangle + \langle V,A^{\sigma}\rangle\right)\langle N,A\rangle + \langle V,A\rangle\langle N,A^{\sigma}\rangle) \\
\frac{\partial d(s,\mathbf{P}_i)}{\partial \tau} &= \frac{(\langle V^{\tau},V\rangle - \langle V,A\rangle(\langle V^{\tau},A\rangle + \langle V,A^{\tau}\rangle))(1-\langle N,A\rangle^2) - |V\times A|^2\langle N,A\rangle\langle N,A^{\tau}\rangle}{SQRT} \\
&\quad -SIGN\left(\langle V^{\tau},A\rangle + \langle V,A^{\tau}\rangle\right)\langle N,A\rangle + \langle V,A\rangle\langle N,A^{\tau}\rangle) \\
\frac{\partial d(s,\mathbf{P}_i)}{\partial \rho} &= \frac{(\langle V^{\rho},V\rangle - \langle V,A\rangle\langle V^{\rho},A\rangle)(1-\langle N,A\rangle^2)}{SQRT} - SIGN\langle V^{\rho},A\rangle\langle N,A\rangle \\
\frac{\partial d(s,\mathbf{P}_i)}{\partial R} &= \frac{(\langle V^{R},V\rangle - \langle V,A\rangle\langle V^{R},A\rangle)(1-\langle N,A\rangle^2)}{SQRT} - SIGN\langle V^{R},A\rangle\langle N,A\rangle
\end{aligned}
$$

The computation of the Hessian matrix is extremely difficult and the evaluation of the 21 items of this matrix would be very time consuming. Therefore we use the non monotone Levenberg-Marquardt minimization method (NMLM) [46], which supplies for the solution vector correction an approximation of the Hessian with the gradient. The analysis of NMLM can be found in [46]. Zhang showed that NMLM converges also linearly, but needs essentially less iterations that the usual monotone Levenberg-Marquardt method.

5.4.1 Obtaining the cone parameterization from the segmentation

In the previous section we mentioned, that we use only the gradient of the distance function, because the Hessian is very hard to compute. Our test have shown, that the

running time of the minimization strongly depends on the provided parameter vector $s(\varphi, \vartheta, \sigma, \tau, \rho, R)$ due to the linear convergence of the Levenberg-Marquardt method. For good first estimation of s the procedure needs only a few iterations to find the minimum. Bad estimations cause very long running time for finding the global minimum of the distance function. Therefore we try to provide the first estimation of the solution vector s as good as possible.

A cone can uniquely be defined by the following three parameters: the parametric equation of the axis, the cone apex and the cone angle (the angle between the axis and the cone mantle). As we will show in the next, these parameters can be derived from our segment and point information and also allow computation of the first estimation of the cone parameter vector $s(\varphi, \vartheta, \sigma, \tau, \rho, R)$.

At first we compute the normal vector of the best fit plane of the end points of the cone normal vectors and assign it as the cone axis vector $A(\sigma, \tau)$, analogously as for the cylinder. The computation of an arbitrary point P_A on the cone axis is illustrated on the fig. 5.5: we choose an arbitrary plane E with the normal vector A and project orthogonally all segment points P_i and their normal vectors $N(P_i)$ to E. We denote P_i^p and $N^p(P_i)$ the projected points resp. the normal vectors. A set of lines $L(P_i)$ going through P_i^p in the direction $N^p(P_i)$ is created. We perform $\frac{n_s}{2}$ line intersections and the centre of mass of the intersection points is assigned to P_A.

The test have shown that a fast and reliable intersection scheme is $L(P_i) \cap L(P_{i+\frac{n_s}{4}})$. The segments were created using the region growing algorithm. Therefore two adjacent points in our data structure are probably also adjacent in the 3D space. If we assume regular growing of the cone segment, the point $P_{i+\frac{n_s}{4}}$ lies shifted about a quarter of the cone mantle from the point P_i.

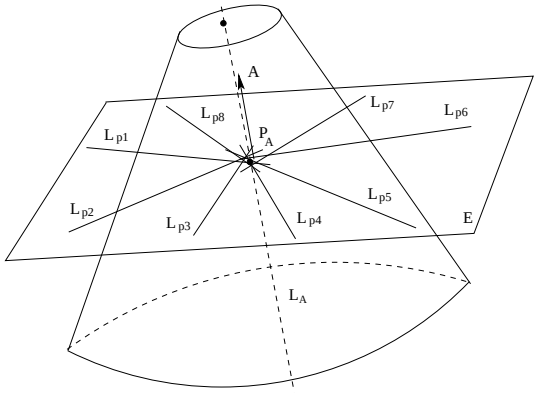


Figure 5.5: Computing of a point of the cone's axis

In the next, the cone axis vector A has to be oriented correctly: we find the points $P_{\min}$ and $P_{\max}$ with the smallest resp. greatest distance to the cone axis. If $\langle A, P_{\min} - P_{\max} \rangle$ is greater than zero, A is oriented correctly, otherwise A is inverted.

To find the cone apex C we compute the centre of mass of n_s intersections of cone mantle tangent lines (with the tangent vector $T = (N(P_i) \times A) \times N(P_i)$) with the cone axis L_A, as shown in the fig. 5.6. Note that the lines do not have to intersect: in that case

if the distance between the lines is smaller than a prescribed tolerance, we project L_2 (see fig. 5.7) to a plane E which is determined by a normal vector $N = N(L_1) \times N(L_2)$ and a point on L_1. We compute the intersection I_1 of L_1 and projected L_2 in the plane E, project this intersection to L_2 and compute the mean of these two points $I = \frac{I_1+I_2}{2}$.

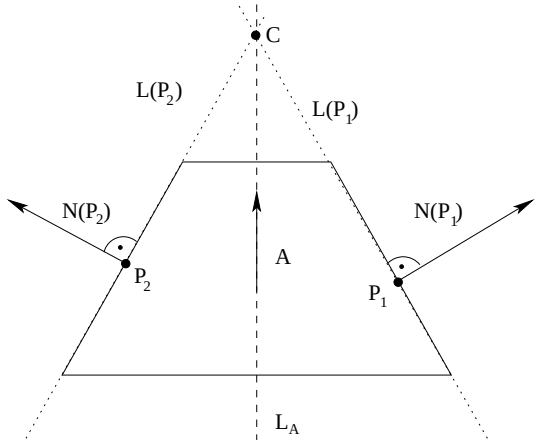


Figure 5.6: Computing the cone apex C

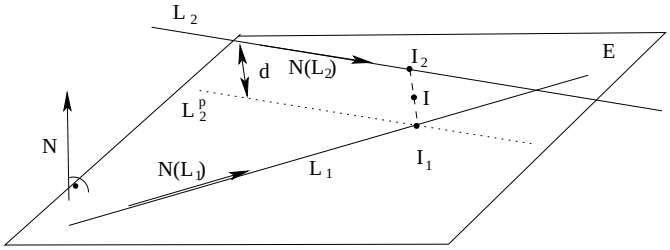


Figure 5.7: Approximate intersection of two lines L_1, L_2

The cone angle can be computed easily from the normal vectors of the points:

$$\alpha = \frac{\pi}{2} - \frac{1}{n_s}\sum_{i=1}^{n_s} \min(\arccos\langle A, N(P_i)\rangle, \pi - \arccos\langle A, N(P_i)\rangle)$$

The vector $N(\varphi, \vartheta)$ from the origin to the nearest point on the cone surface can be now computed according to the fig. 5.8:

$$\begin{aligned} N' &= (F-O) + w(-A) \qquad \text{where} \\ w &= \|F-O\| \tan\alpha, \\ F &= C - \langle C, A\rangle A, \qquad \text{so} \end{aligned}$$

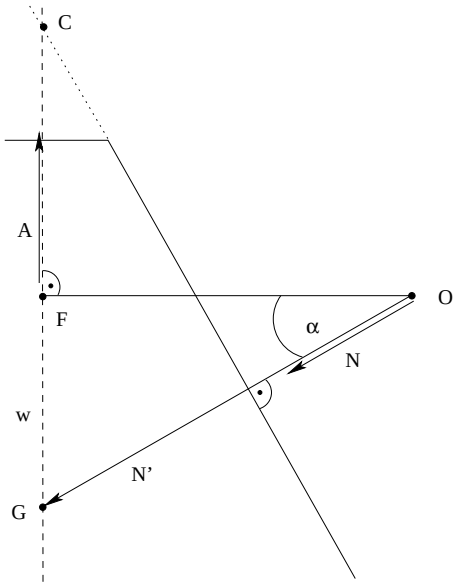


Figure 5.8: Computing $N(\varphi,\vartheta)$

$$N' = C - A(\langle C,A\rangle + \|C - \langle C,A\rangle A\| \tan\alpha) \qquad \text{and}$$

$$N = \frac{N'}{\|N'\|}$$

Note that a special case when the origin lies on the cone axis has to be handled separately: we can choose the direction of $N(\varphi,\vartheta)$ arbitrarily[2], e.g.:

$$N' = (\text{drand48 }(),\text{drand48 }(),\text{drand48 }()) \times A$$
$$N = -\left(A\cos(\frac{\pi}{2}-\alpha) + \frac{N'}{\|N'\|}\sin(\frac{\pi}{2}-\alpha)\right)$$

Note, that the Unix function 'drand48 ()' generates a random number in the interval [0,1].

Let G be the intersection point of the cone axis and the line from origin in the direction of $N(\varphi,\vartheta)$, see fig. 5.8. Then $R = \|C-G\|\sin\alpha$. If the intersection point lies behind the cone apex (looking from cone interior, i.e. $\langle G-C,A\rangle > 0$), the radius has to be inverted: $R = -R$. Finally: $\rho = \|G-O\| - R = \|G\| - R$.

[2]Under condition $\angle(-N,A) = \frac{\pi}{2} - \alpha$

5.5 Extension procedure

If a segment S is recognized as one of the four basic types, the neighboring points of this segment which fulfill the given surface equation are added to S, i.e. if P is a point of the segment S and $Q \in \mathcal{N}(P)$, but $Q \notin S$, then Q is integrated into S if the distance of Q to the fitted surface is less than a prescribed threshold. Let Q be the point which is going to be added to a segment, then Q will be inserted to a

- **planar segment** S_p, iff the orthogonal distance of Q to the best fit plane of S_p stays smaller than a threshold T_p.
- **spherical segment** S_s with the sphere centre C and a radius R, iff $\big|\, \|C-Q\| - R| < T_s$.
- **cylindrical segment** S_c with the axis A, A_p (A is the axis vector, A_p is a point on the axis) and radius R, iff $\big|\, \|Q^p - A_p\| - R| < T_c$, where Q^p is the orthogonal projection of Q into a plane determined by A, A_p.
- **conical segment** S_o with the axis vector A, cone apex C and cone angle α, iff $\|Q-C\| \sin|\omega - \alpha| < T_o$, where $\omega = \arccos \frac{\langle Q-C, -A\rangle}{\|Q-C\|}$, see fig. 5.9.

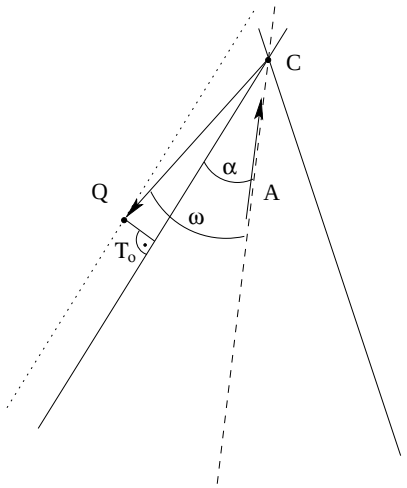


Figure 5.9: A distance of Q to the cone mantle

The estimation of the thresholds T_p, T_s, T_c and T_o will be explained in detail in the section 5.7.

The fig. 4.11 b) in the section 4.3.1 illustrated the second order segmentation of a noisy object. Due to the noise and decent but not perfect estimation of the segmentation thresholds the basic surfaces of the object (planes, spheres, ...) were segmented into many segments, e.g. the conical part consists of ca. 15 segments. The main goal after

surface fitting is to represent such an object's part with one surface. Processing all segments and joining two neighboring surfaces the with the same or similar geometric properties turned out as very unreliable because the small segments have been often fitted to wrong surface types (e.g. a small part of a cylindrical or conical segment can be recognized as a noisy planar segment).

The most natural and stable process is to recognize the big segments and let them grow. Therefore for a segment S, that is being extended, the distance of $Q \notin S$ to the surface is computed. If this distance is less than a threshold and

- the point Q is not marked[3], Q is added to the segment S.
- the point Q is marked, the segment S_Q containing Q is found and
  - if S_Q is **not** flagged as *processed segment*, "steal" Q from S_Q and add it to S.
  - otherwise find the next neighbor $Q \notin S$.

In order not to "steal" points from already processed big segments, we introduced a flag: if the number of points of an extended segment is greater than a threshold T_n, the segment is marked as *processed* and no point can be stolen from this segment. Note that T_n is derived from the whole number of points and number of big segment after second order segmentation, see section 5.7. In special case the user can decide not to mark the processed segments. The explanation of such a case with an example can be found at the end of the next section.

During the segmentation only a small part of the surface can be detected as a segment and the fitting procedure can provide rough surfaces parameters. Inserting many new points into a classified segment often affects the surface parameters. It is obvious, that the bigger is the segment, the more exact the surface parameters are approximated. Therefore, if the ratio

$$\frac{\text{Number of points after extension}}{\text{Number of points before extension}} > \lambda$$

holds, then we recompute the algebraic equation of the surface (i.e. we repeat the surface fitting) based on the extended point set. The process of extension and recomputing is repeated until the ratio is less than λ.

Note that we set $\lambda = 1.3$ (increasing of the segment size about more than 30%) constantly.

5.6 Postprocessing procedure

The postprocessing consists of the steps of segment classification and segment extension. The classification makes an assumption about the segment's type based on predefined thresholds and starts the minimization (fitting) procedure. During minimization the sum of the distances of the points to the surface is computed. The average distance

[3]The points of big segments are marked, the other points stay unmarked

is used to reject the segment if an incorrect assumption was made. If the segment is accepted the extension procedure tries to add new neighboring points which fulfill the given surface's equation.

The general problem of the classification procedure are the thresholds which decide if a segment will be accepted for the minimization procedure or not. It is impossible to provide reliable thresholds for all object types automatically, even the user interaction may fail in many cases: as an example we can mention a planar segment which can be classified as a cylindrical one with a huge radius, or a thin cylindrical slice can be recognized as a part of a sphere, see fig. 5.10.

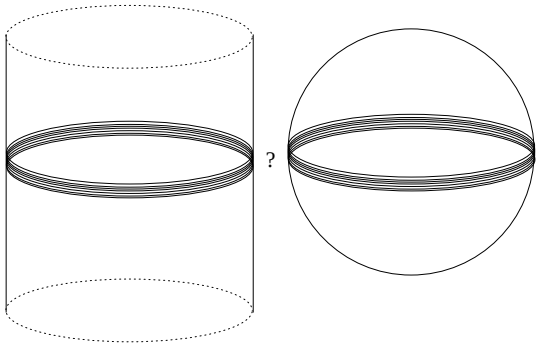


Figure 5.10: Cylindrical or spherical segment?

Therefore the postprocessing procedure should be able to perform successive improvements of the existing segmentation and classification. Our implementation of the postprocessing procedure allows many passes, independent on the number of already classified segments. For example if an object was processed with insufficient thresholds and many segments stayed unclassified, the postprocessing procedure can be started again with different thresholds and the main goal of the postprocessing is

- to improve the shape of already classified segments and extend them if possible,
- to reject classified segments (according to the average distance criterion), if they were recognized incorrectly in the previous pass,
- to recognize and extend the remaining segments.

The overall structure of the postprocessing procedure is as follows:

1. Discard small segments.
2. Mark points of unreleased segments.
3. Reevaluate recognized segments; accept / reject a segment based on point distance criterion D_A. Rejected segments are released and their points are unmarked.[*]

4. Perform quick curvature based segmentation on unmarked points with modified segmentation parameters.

5. Discard small segments again and unmark their points.

6. Sort the segments according to the ascending sum of the reference curvatures $\kappa_{\min}^{\text{ref}} + \kappa_{\max}^{\text{ref}}$. A segment list L_1 results.

7. Apply plane detection (with subsequent extension) on the segments from L_1 until *plane criterion* fails.

8. Sort remaining segments according either the average distance of the segment's points to the fitted surface [*] or their size. A segment list L_2 results.

9. Take S_i from L_2, $i = 1, \ldots, n_l$.

10. **If** S_i has been classified **then** repeat the fitting; start the extension.[*]

11. **Else** compute the best fit plane B_E of the end points of the normal vectors projected to Gauss sphere.

12. **If** $\frac{1}{n_s}\sum_{i=1}^{n_s} d(\mathbf{P}_i, B_E) < T_{c2}$ **then** fit S_i to cylinder / cone and start the extension.

13. **Else If** $\kappa_{\max}^{\text{ref}} - \kappa_{\min}^{\text{ref}} < T_{s2}$ **then** run sphere fitting for S_i and thereafter start the extension.

14. **Else** Let S_i unclassified and continue at 9.

Note that the steps (resp. the part of the step) marked with '[*]' are relevant only for the second (and the next) pass of the postprocessing procedure. During the first pass, these steps are skipped.

One would assume that it is possible to recognize planes based on a simple test of the reference curvature values: $\kappa_{\min}^{ref} + \kappa_{\max}^{ref} < \varepsilon$. However, this approach turned out to be unreliable because of the difficulty to provide a good choice for the threshold ε. Instead we based our detection of planar segments on the computation of a best fitting plane (step 7 of the postprocessing procedure) and a test whether the points in the segment stay close to this plane. Note, that the threshold T_p needed for this test can easily be adjusted to the size of the object and the percentage of noise expected in the data.

Since we want to avoid to fit planes to all segments the following operations are performed. First, segments with a number of points less than a prescribed value MIN_SEGMENT (default 40) are marked as small and are disregarded for surface classification. The remaining segments are ordered according to ascending values $\Delta\kappa_{\min}^{ref} + \Delta\kappa_{\max}^{ref}$ (step 6). The plane detection starts with the first element of the ordered segment list and continues until the second segment is found, which cannot be classified as a plane: it turned out to be more stable for noisy segments, if the first non planar segment in the list is found, to continue testing for planar segments until the second non planar occurs. Thereafter we swap the first non planar with the last

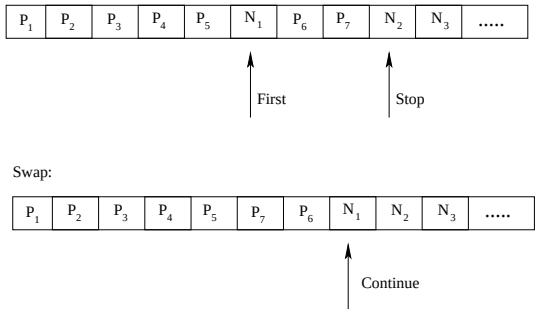


Figure 5.11: Planar segments detection

planar segment and proceed testing from the first non planar for next surface types, see fig. 5.11.

The remaining elements in the ordered segment list are subject to the continuing surface classification. The step 8 of the postprocessing structure shows that it is advantageous to classify the remaining segments according to their average distance of the points to the fitted surface (for already classified segments) or in descending order of their size (number of points) beginning with the largest one.

An already classified segment has a higher priority than an unclassified one (even bigger) and therefore the classified segments are processed before the unclassified ones. A segment after classification is extended (section 5.5): if the surface parameters were well estimated in the previous pass, the average distance (D_A) of the points to the surface stayed small and in the current pass the segment is processed among the first ones. If the surface parameters were estimated insufficiently, the extension behaves unpredictable and the average distance D_A will grow. In the current pass such a segment is verified - step 3 - and if $D_A > T_d$, the segment is marked as an unclassified one. If $D_A < T_d$ but still big, the segment is placed at the end of the list of the classified segments and the probability, that the well classified segments will "steal" incorrectly added points from this segment, is high. In the fig. 5.12 such a case is demonstrated: because of inappropriate thresholds the sphere and the cone grew too far inside of the cylindrical surface. In the second pass the points in the dotted areas of the classified conical and spherical surface cause a large average distance D_A and the cylinder will be placed in the sorted list before them. During the extension the cylinder will "steal" all the points on the cylindrical part of the object from the sphere and cone and thereafter the cylindrical segment is marked as *processed*, i.e. the cone and sphere cannot steal any point from it, even if they would be extended with the same thresholds as in the first pass.

The step 4 of the postprocessing hierarchy solves problems caused by inappropriate segmentation thresholds. To provide correct segmentation parameters for noisy objects with many C^1 transitions is extremely difficult. Also the user, inspecting the results of the segmentation and its postprocessing, often is not able to hit the perfect segmentation parameters:

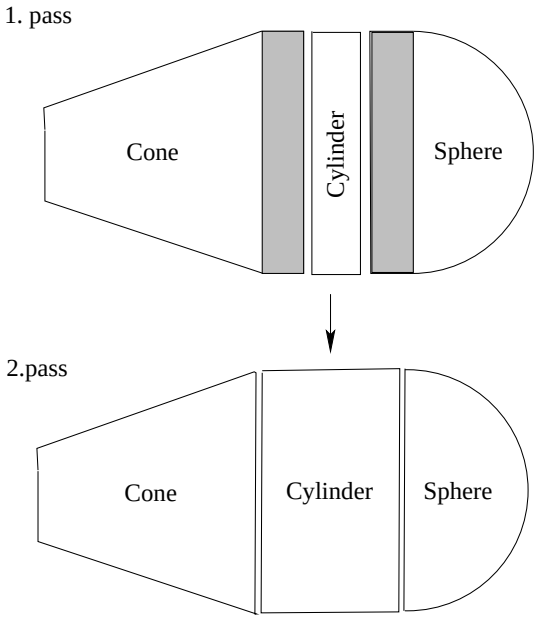


Figure 5.12: Points "stealing"

- increasing the thresholds can cause joining two distinct geometric surfaces with C^1 transition between them.
- decreasing the thresholds can cause neglecting some small (but important) segments, for an example see fig. 5.13 **a)**.

We prefer the second case: to subdivide the object into separate segments, that correspond to the big geometric surfaces and accept that small surfaces will not be found rather than to find all surfaces with vague C^1 transitions.

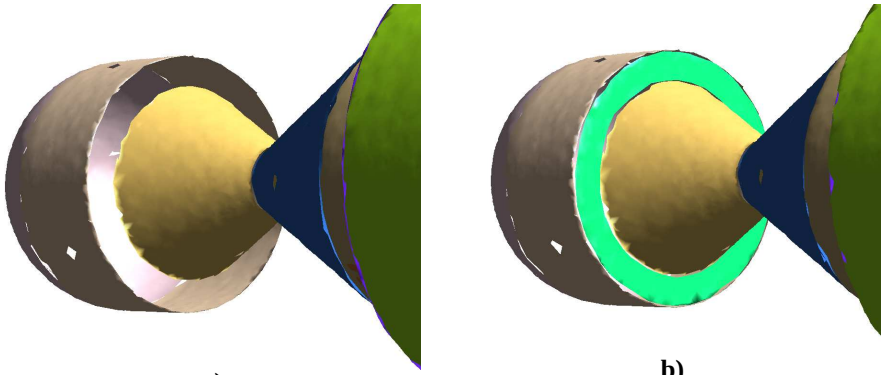
During the postprocessing small segments and the segments, which were recognized incorrectly in the previous postprocessing pass, are discarded. The released points are put into the segmentation procedure with modified segmentation parameters: $\lambda_n\Delta\kappa_{\min}$, $\lambda_n\Delta\kappa_{\max}$, $\lambda_n\Delta\kappa^{ref}_{\min}$, $\lambda_n\Delta\kappa^{ref}_{\max}$ (the multiplication coefficient λ_n is taken default 1.1, but the user can adjust it). Small, during the segmentation not found, geometric surfaces or incorrectly recognized surfaces (and thus incorrectly extended surfaces) in the previous postprocessing pass are now segmented again and added to the list of segments to be processed. Note, that the repeated segmentation cannot steal points from the regular segments.

The repeated segmentation works on a reduced point set and is time optimized, so it takes a few tenths of a second for a data set of ca. 100,000 points.

In the fig. 5.13 **a)** the small plane between the cylinder and the cone was not detected due to the small segmentation thresholds. After first postprocessing pass all other geometric surfaces were recognized correctly. The user had to start the postprocessing

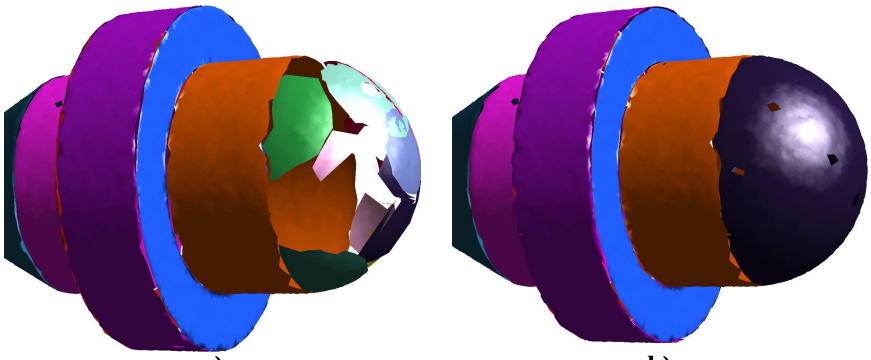
procedure again with the only change: increase the multiplication factor λ_n to 1.3.

In the fig. 5.14 **a)** small thresholds caused subdivision of the sphere into many "almost planar" segments. After inspecting the result the user had to shrink slightly the plane detection threshold and increase λ_n to e.g. 1.3. In the next postprocessing pass the planes on the sphere were discarded, their points put into the segmentation procedure, which created a spherical segment, that was correctly recognized and extended, see fig. 5.14 **b)**.



a) **b)**

Figure 5.13: Segmentation and recognition problems with a small plane



a) **b)**

Figure 5.14: Segmentation and recognition problems on a C^1 transition

As next we perform a test for a cylinder or cone. We use a stable and object size independent test condition and hence the probability for a wrong assumption is lower than for a sphere, which is processed hereafter.

Let S denote a segment that undergoes the test for a conical or cylindrical surface. First, the Gaussian image of S is computed, i.e. the point set $\mathcal{G}(S)$ on the unit sphere is

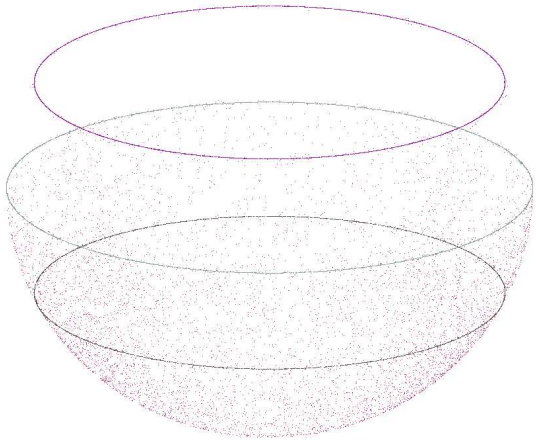


Figure 5.15: Gaussian Image of one cylindrical, two conical and one spherical segment

created that corresponds to the endpoints of all normal vectors of S, see fig. 5.15. Then, the best fitting plane E of $\mathcal{G}(S)$ is computed. If all points of $\mathcal{G}(S)$ have a distance to E that is smaller than a tolerance T_{c2}, S is considered to be conical or cylindrical. (Note, that T_{c2} is independent of the size of the object, as the Gaussian sphere has a constant radius. We use $T_{c2} = 0.02$ as a default value). If E contains the center of the unit sphere, S is part of a cylinder, otherwise cone. Note that planar segments, which would also fulfill the T_{c2} criterion were processed in the beginning and then removed from the list of segments to be examined.

To prevent the postprocessing procedure from extending a wrong classified segment, we verify the fitted segment, if it fulfills "correct" surface properties, e.g. a cylinder with a big radius can be interpreted as a plane, or a cone with a small cone angle as a cylinder. In order to accept a cylinder, its radius has to be smaller than a threshold T_r. In order to accept a cone, the cone angle has to be in the interval $(T_{\alpha 1}, T_{\alpha 2})$, for details see the section 5.7. Otherwise it could be unreliable to continue with the segment extension.

At the end the spheres are processed: let S be a segment to be classified. If the value $\Delta = \kappa_{\max}^{\text{ref}} - \kappa_{\max}^{\text{ref}}$ of S is smaller than a threshold T_s, we try to fit S by a spherical segment. Note that we regard the difference of the reference curvatures, which should be the same. Thus T_{s2} should be objects size independent. But the test have shown, that T_{s2} slightly depend on the object's size and can be derived from e.g. the bounding box of the object.

In the section 5.5 we mentioned that in special cases the flag "processed segment" can be switched off. This involves cases when on a cylindrical or conical segment a big planar segment is created (strips along the axis) and marked as processed. To decrease the plane detection threshold would not solve this situation because regular planar segments would not be recognized or they could be incorrectly recognized as cylinders or cones. Instead of modifying the plane threshold the user has to switch off marking processed segments. The condition for this action is the existence (recognition) of at

least one big conical or cylindrical segment on the cylindrical / conical surface. During the extension this segment can grow and steal points from all segments regardless their size. The disadvantage is that on C^1 transitions the points can be assigned many times to a segment (they can be stolen many times), which increases the processing time. The second flaw is the user assistance - at present we are not able to detect such cases automatically.

In the figure 5.16 such a case is illustrated, when after inspecting the results the user has to decide to turn off marking of the processed big segments.

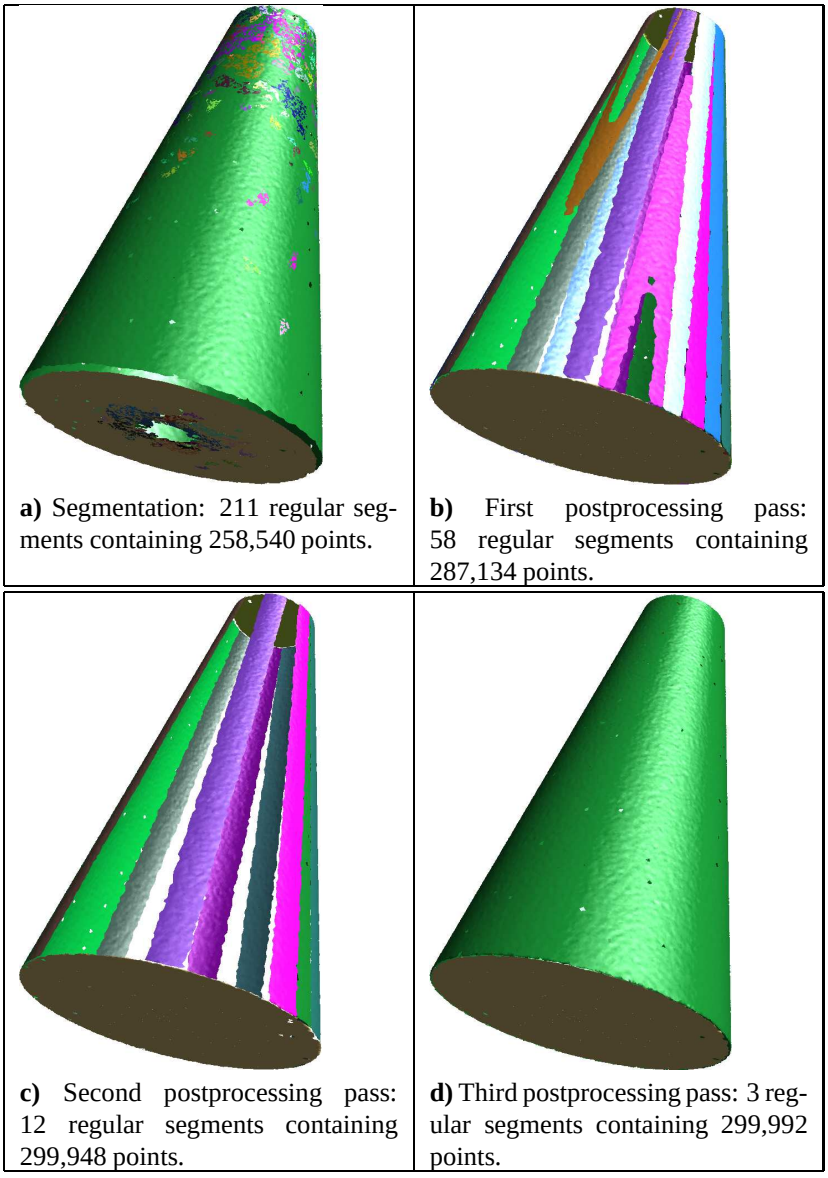
We generated a simple object (cone) consisting of a few surfaces (3 surfaces) and many points (300,000 points) strongly perturbed by noise. Due to the noise the segmentation was not able to detect the cone mantle as one conical segment and therefore many segments on the mantle resulted - figure **a)**. During the first postprocessing pass some of these segments were incorrectly recognized as planes and they were extended (strips on the cone mantle) - figure **b)**. In this pass no conical segments was recognized and therefore we had to increase the cylinder/cone detection threshold. After the second postprocessing pass beside the planar segments on the cone, two big conical segments were detected (containing 104,752 and 39,866 points) - figure **c)**. Further postprocessing steps with growing extension thresholds would not bring any improvements, because the planar segments are processed in the beginning of the postprocessing and because of their big size they are marked as processed. Switching off marking of the processed segments yields the desired result - figure **d)**.

Note that only points in the regular segments are counted. Remaining points have not been assigned to any segment. Improved approximation of the surface parameters of geometric surfaces forming the object in the particular postprocessing steps causes growing number of points in the regular segments and decreasing of the number of regular segments.

The whole curvature based segmentation procedure (with subsequent postprocessing and classification) is time optimized, in order to allow many iterations and tuning of the postprocessing parameters. Table 5.1 shows the time consumption (in seconds) of the segmentation and postprocessing of a few objects (run on a Athlon 1200 MHz system, with 512 MB DDR-RAM). Note that the column *I. order segmentation time* involves the computation of the k-nearest neighbors, normal vector estimation, consistent orientation of the normal vectors and three repetitions of segmentation and normal vector recomputing.

The time complexity depends mainly on the number of geometric surfaces in the object rather than on the number of points. The Levenberg-Marquardt minimization method converges linearly (in contrast to the quadratic convergence of the Newton-Raphson method), hence many cones cause bigger time consumption than cylinders or spheres. The tests have shown that the minimization procedure of already fitted and extended segments needs in general less iterations than the minimization from the segmentation parameters and therefore the second (and the other) postprocessing passes run faster than the first one.

Note, that if an object consists only of free form surfaces, the postprocessing can be switched off by the user.



a) Segmentation: 211 regular segments containing 258,540 points.

b) First postprocessing pass: 58 regular segments containing 287,134 points.

c) Second postprocessing pass: 12 regular segments containing 299,948 points.

d) Third postprocessing pass: 3 regular segments containing 299,992 points.

Figure 5.16: Reconstruction of a noisy cone

The figures 5.17– 5.23 illustrate the segmentation and the postprocessing on an artificially created point set containing 90,974 points. The figures 5.17– 5.19 show the whole process applied on the original data set. The data set in the figures 5.20– 5.23 was contaminated with random noise (0.1% of the diagonal length of the object's bounding box). For all test the default segmentation thresholds were used.

The noiseless data set could be processed with default postprocessing parameters,

# of points	# of surfaces	Noise amount estimation	I. order segment. time	Curvature estimation time	II. order segment. time	Postproces. time
22,211	22	50%	5.04	0.97	0.35	2.45
54,854	11	60%	12.25	2.32	1.26	4.25
90,974	18	60%	19.28	3.88	2.53	7.39
300,000	3	90%	77.95	13.81	4.64	12.54

Table 5.1: Segmentation and postprocessing (including the classification) time consumption of some objects. All times are in seconds.

no user iteration was necessary.

In the beginning of the processing of the noisy data set, the user specifies his subjective estimation of the noise (about 60%). After first postprocessing pass only the sphere staid unrecognized, see fig. 5.22. Increasing the sphere detection threshold results in completely segmented and classified object. Note that the boundary of the cylinder (between the block and the cone, see fig. 5.23) cannot be detected correctly, as because of the noise all the postprocessing thresholds had to be set higher and thus the points on the planes also fulfill the algebraic cylinder equation.

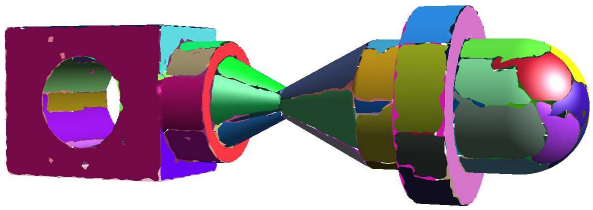


Figure 5.17: The first order segmentation: 76 regular segments containing 89,753 points (of 90,974).

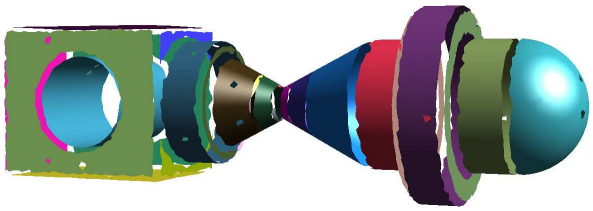


Figure 5.18: The second order segmentation: 25 regular segments containing 76,343 points (of 90,974).

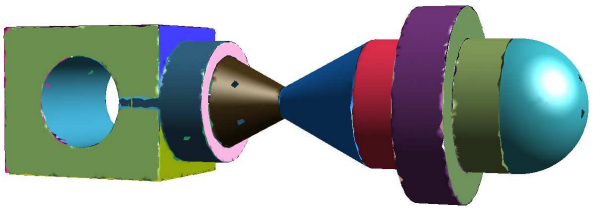


Figure 5.19: The first postprocessing pass: 18 regular segments containing 90,963 points (of 90,974).

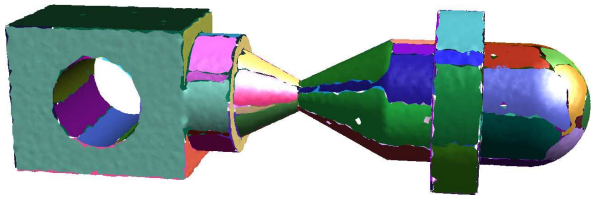


Figure 5.20: The first order segmentation: 77 regular segments containing 88,877 points (of 90,974).

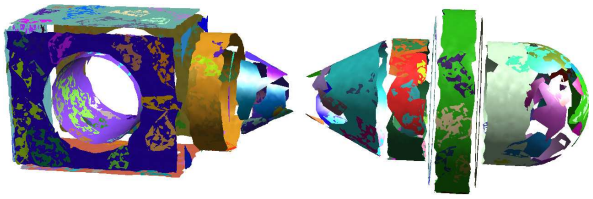


Figure 5.21: The second order segmentation: 171 regular segments containing 46,793 points (of 90,974).

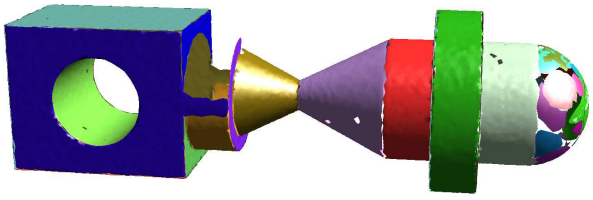


Figure 5.22: The first postprocessing pass: 37 regular segments containing 87,814 points (of 90,974).

5.7 Postprocessing thresholds

In the previous sections we introduced many thresholds necessary for a successful postprocessing. In this section we conclude the functionality of all thresholds and show how to choose them, resp. how a few of them can be replaced by one global threshold.

Till now we used the following postprocessing thresholds:

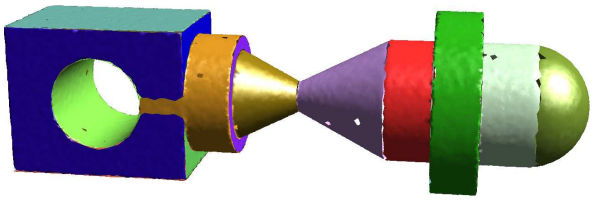


Figure 5.23: The second postprocessing pass: 18 regular segments containing 90,898 points (of 90,974).

T_p	Detection and extension of a plane
T_{s2}	Detection of a sphere
T_s	Extension of the fitted sphere
T_{c2}	Cone / Cylinder detection
T_c	Extension of the fitted cylinder
T_o	Extension of the fitted cone
T_d	Distance threshold for rejecting / accepting fitted segments
T_n	Minimal size for the flag *processed segment*
T_r	Maximum radius for a sphere or a cylinder
$T_{\alpha 1}, T_{\alpha 2}$	Minimum / maximum cone angle

Table 5.2: Postprocessing thresholds

Let $d_d(\mathcal{B})$ denote the diagonal length of the object's bounding box, $d_s(\mathcal{B})$ the shortest side of the bounding box.

The last three thresholds prevents the postprocessing from wrong surface assumption and fitting: a cylinder can be confused with a cone with a small cone angle; a plane can be confused with a cone with big angle or with a cylinder (or sphere) with a big radius. The experiments have shown, that $T_{\alpha 1}$ and $T_{\alpha 2}$ can be chosen statically as $T_{\alpha 1} = 5°$ and $T_{\alpha 2} = 80°$, but T_r has to be scaled according to the object's bounding box size: $T_r = \lambda_r d_s(\mathcal{B})$. λ_r is specified empirically.

After the second order segmentation a huge number of segments result. We release small segments and the remaining ones are regarded for the following postprocessing. If n is the total number of points in the point set and m is the number of remaining segments, then we set $T_n = \frac{n}{m+\lambda_n}$, where $\lambda_n \approx 5$, as the threshold for the flag "processed segment". This threshold approximates the average number of the points of the segments, which are going to be processed.

The threshold T_d controls accepting/rejecting a segment after fitting, i.e. the allowed maximum for the average distance of the points to the fitted surface. Therefore T_d coheres with the size of the object and can be derived e.g. $T_d = \lambda_d d_s(\mathcal{B})$ (λ_d is specified empirically).

At the end of the section 5.6 we mentioned that the cylinder detection threshold does not depend on the object's size and therefore can be set constant, e.g. $T_{c2} = 0.2$. The test have shown that the user only seldom has to modify (increase) the threshold, in most cases the default setting guarantees satisfactory detection.

One would assume that the difference of the sphere reference curvatures should be independent on the object's size. We observed, that with growing reference curvatures, the difference is also greater and the threshold also has to grow. Thus the sphere detection threshold T_{s2} is indirect proportional to the object's size: the smaller is the sphere the higher are the principal curvatures on the sphere surface and the greater can be the difference $\kappa_{\max}^{\text{ref}} - \kappa_{\min}^{\text{ref}}$. We used the bounding box diagonal length $d_d(\mathcal{B})$ to estimate the sphere detection threshold.

The remaining thresholds T_p, T_s, T_c and T_o specify the maximum distance of a new point, which is going to be added to the current segment in the extension procedure, to the fitted surface. All these thresholds measure the same absolute distance and that is why we replaced them by one "global" threshold T_g, which can be again derived from the object's bounding box size. In all our tests T_g provided reliable extension. In some special cases the user can switch off T_g and specify T_p, T_s, T_c and T_o separately.

At the end of our automatic threshold estimation procedure the thresholds T_g, T_{s2}, T_{c2} and T_d are adjusted according to the user specified noise amount in the data.

Chapter 6

Results

In this section we present results obtained by the second order segmentation of real objects or artificially generated objects contaminated with random noise (always 0.1% of the diagonal length of the object's bounding box). For all objects at most 2 postprocessing passes had to be performed.

For the figure 6.1 we use the following figure notation $\begin{array}{c|c|c} a) & b) & c) \\ \hline d) & e) & f) \end{array}$. The object **a)** was the only point set, where the sphere detection threshold had to be modified (the other objects were reconstructed successfully with default detection thresholds). The sphere has the radius $r = 2$ and the object was contaminated with 0.021 noise, i.e. 0.5% of the sphere size. The C^1 transition between the sphere and the cylinder causes a "floating" border between the cylinder and sphere depending on the processing order, i.e. if the sphere is processed first, it takes a small part of the cylinder and vice versa.

The objects **b) – d)** needed only one postprocessing pass with default thresholds. The object **e)** is very hard to reconstruct if the noise amount exceeds 0.2% because of many C^1 transitions.

During the postprocessing of the object **f)** the case depicted in the fig. 5.10 occurred, i.e. the thin cylindrical strip between the two planar disks was classified as a spherical segment. This ambiguity can be for this case solved, if the user switch off sphere detection before the postprocessing: the result together with the segments list is illustrated in the fig. 6.3.

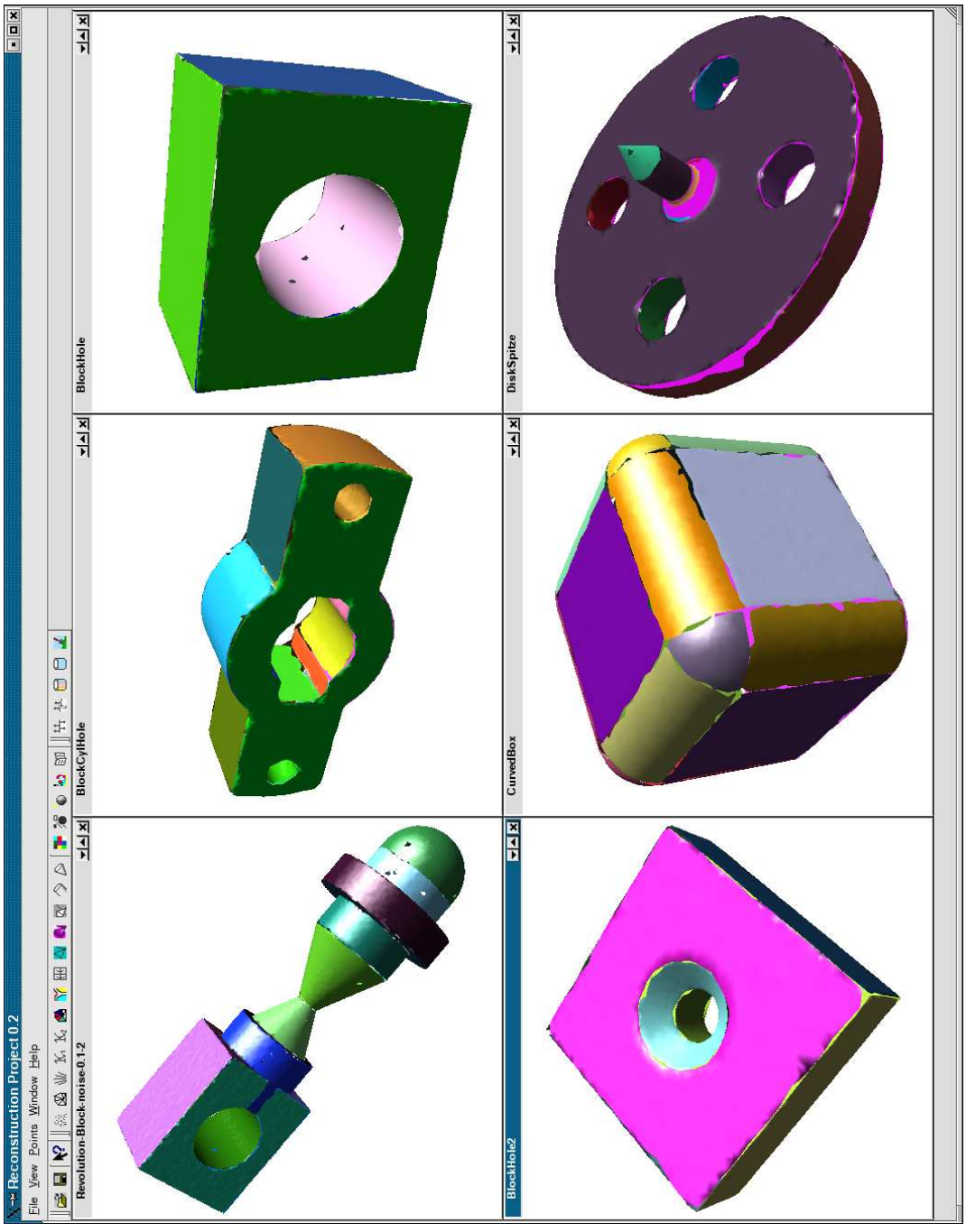


Figure 6.1: Results 1

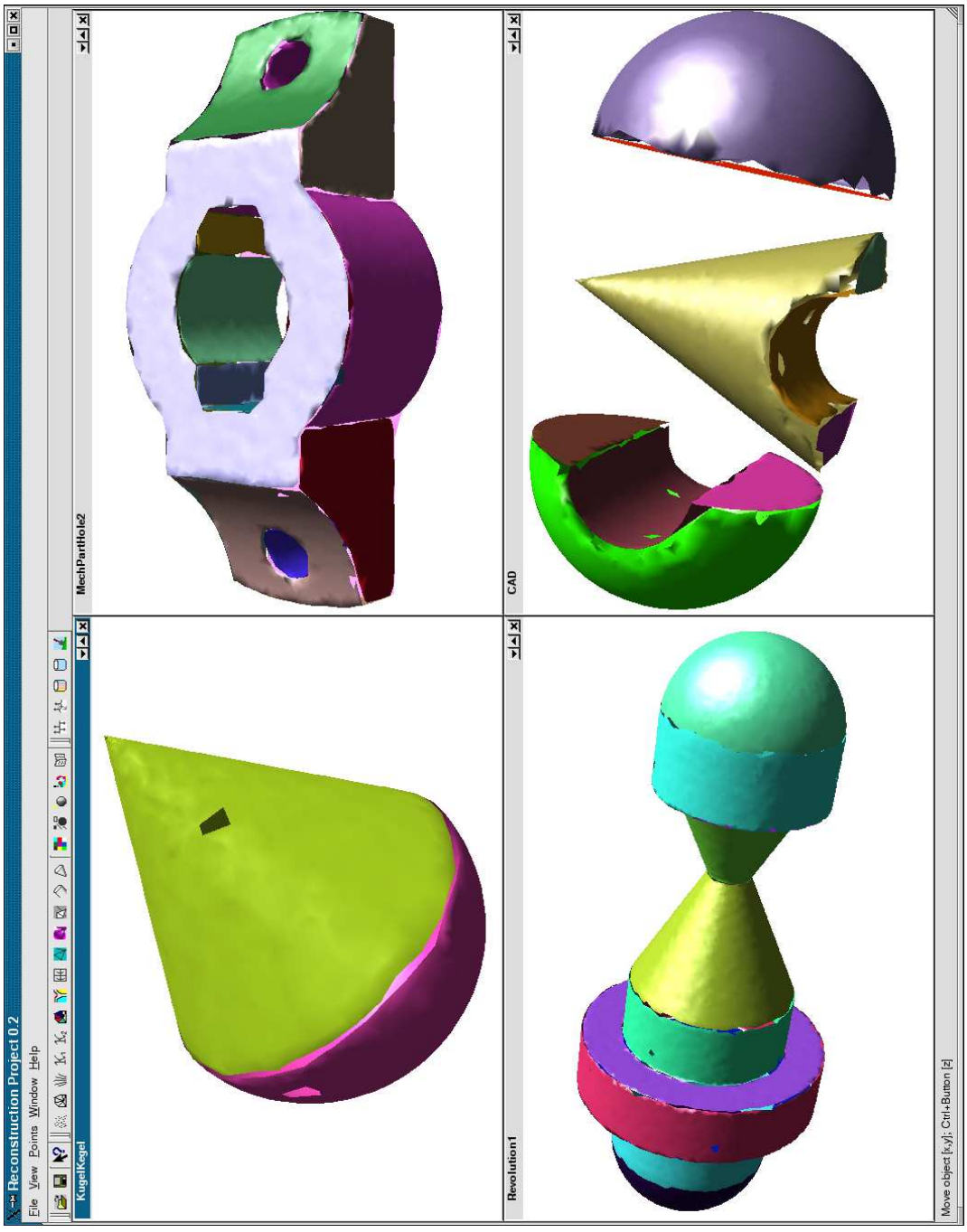


Figure 6.2: Results 2

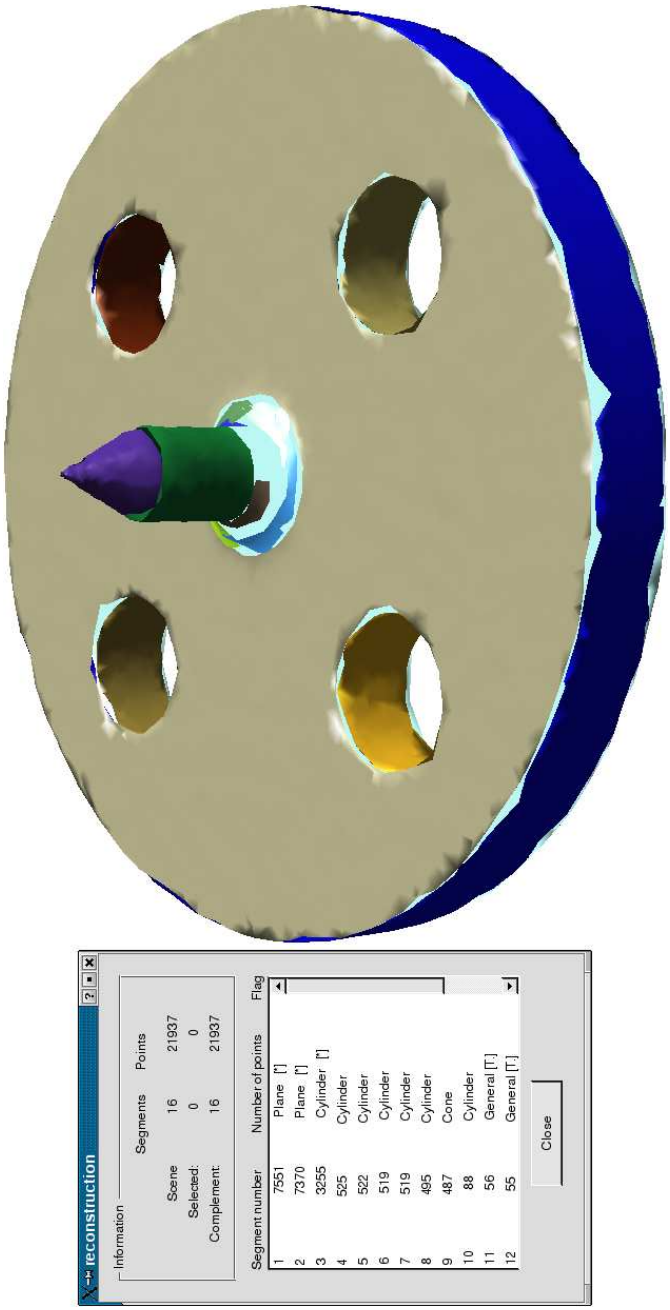


Figure 6.3: The object from the fig. 6.1 **f)**, if the sphere detection was switched off.

Chapter 7

Conclusion

We presented an approach for automatic segmentation and recognition of digitized 3d objects. We expect only unorganized boundary points of a 3d object and do not require any further surface information. For a successful reconstruction an appropriate sampling is expected, but we do not necessarily need an uniform sampling.

We developed stable procedures, which provide from the given points surface properties estimations, that are the basis for the first and second order segmentation. The essential parameters for both segmentations are computed and provided to the user by our program, in order to decrease the user assistance.

The most important part of our work - the postprocessing procedure attempts to fit the supplied segments by simple algebraic surfaces and in the case of success, extend the surfaces by adding adjacent points to them. The postprocessing is very robust against the noise - it is able to recognize the algebraic surfaces correctly even in data sets contaminated with strong noise.

All steps of the reconstruction process are time optimized for fine tuning of the segmentation or postprocessing and can be independently repeated in the following order: normal vector estimation - first order segmentation - curvature estimation - second order segmentation - postprocessing, i.e. if e.g. the second order segmentation fails, only the second order segmentation have to be repeated, the first order segmentation, the normal vector and curvature estimation do not have to be started again. If the second order segmentation creates a proper subdivision of the surface, postprocessing can be started. If the postprocessing fails, only the postprocessing should be repeated.

The interface between the user and our procedures is implemented using Qt library and can be very easily ported to any Unix or Windows based platform. At present we ported and compiled the program under Linux, SGI IRIX, HPUX and Windows 2000/XP. In the figures 6.1, 6.2, 6.3 a part of the graphics user interface can be seen.

In the future several extensions of the segmentation and recognition procedures are planned.

Bibliography

[1] **Amenta, N. and Bern, M.:** Surface reconstruction by Voronoi filtering, *Discrete and Computational Geometry*, 1999, 481–504.

[2] **Amenta, N., Choi, S. and Kolluri, R.:** The power crust, unions of balls, and the medial axis transform, *Computational Geometry: Theory and Applications*, 2001, Vol. 19, 127–153.

[3] **Amenta, N., Choi, S. and Kolluri, R.:** The Power Crust, *Proceedings of 6th ACM Symposium on Solid Modeling*, 2001, 249–260.

[4] **Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R. and Wu, A.Y.:** An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimension, *Journal of the ACM 45*, 1998, 891–923.

[5] **Bajaj, Ch., Bernardini, F. and Xu, G.:** Automatic reconstruction of surfaces and scalar fields from 3D scans, *Computer Graphics Proceedings, SIGGRAPH '95, Annual Conference Series*, 1995, 109–118.

[6] **Barequet, G. and Sharir, M.:** Piecewise-linear interpolation between polygonal slices, *Proceedings of the 10th annual symposium on Computational Geometry*, June 1994, Stony Brook, New York, USA, 93–102.

[7] **Bentley, J.L. and Shamos, M.I.:** Divide-and-conquer in multidimensional space, *Proc. of the 8th Annual Symp. on Theory of Complexity*, 1976, 220–230.

[8] **Bergevin, Soucy, Gagnon and Laurendeau:** Towards a general multi-view registration technique, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1996, Vol. 18(8), 540–547.

[9] **Bernardini, F., Mittleman, J., Rushmeier, H., and Silva, C.:**, The Ball-Pivoting Algorithm for Surface Reconstruction, . *IEEE Transactions on Visualization and Computer Graphics*, October-December 1999, Vol. 5, No. 4, 349–359.

[10] **Besl and McKay** A method for registration of 3D shapes, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992, Vol. 14(2), 239–256.

[11] **Boissonnat, J.-D. and Geiger, B.:** Three Dimensional Reconstruction of Complex Shapes Based on the Delaunay Triangulation, *Research report No 1697, INRIA, Sophia Antipolis*, April 1992.

[12] **Boissonnat, J.-D.:** Geometric structures for three dimensional shape representation, *ACM Transactions on Graphics, 3(4)*, 1984, 266–286.

[13] **Boissonnat, J.-D. and Cazals, F.:** Smooth Surface Reconstruction via Natural Neighbour Interpolation of Distance Functions,*ACM Symposium on Computational Geometry*, 2000.

[14] **Brunnett, G., Schädlich, Th., Vančo, M.:** Extending Laszlo's Algorithm to 3D, *Proceedings of the International Conference on Imaging Science, Systems and Technology*, CISST'2001, 499–505.

[15] **Chen and Medioni:** Object modelling by registration of multiple range images, *Image and Vision Computing*, 1992, Vol. 10(3), 145–155.

[16] **Chen, H.-Y., Lee, I.-K., Leopoldseder, S. , Pottmann, H., Randrup, T. and Wallner, J.:**, On Surface Approximation Using Developable Surfaces, *Graphical Models and Image Processing*, Vol 61, 110-124, 1999.

[17] **Devillers, O.:** Improved incremental randomized Delaunay trinagulation, *In Proc. 14th Annu. ACM Symposium on Computational Geometry*, 1998, 106–115.

[18] **Edelsbrunner, H. and Mücke, E.:** Three dimensional alpha shapes, *ACM Transactions on Graphics 13(1)*, 1994, 43–72.

[19] **Eggert, Fitzgibbon and Fisher:** Simultaneous registration of multiple range views for reverse engineering, *International Conference of Pattern Recognition*, 1996, 243–247.

[20] **Faugeras, O.D. and Herbert, M.:** The representation recognition and locating of 3D objects, *International Journal of Robotics Research 5*, 1986, 27–52.

[21] **Fuchs, H., Kedem, Z. M. and Uselton, S. P.:** Optimal Surface Reconstruction from Planar Contours, *Comm. ACM 20*, 1977, 693–702.

[22] **Gopi, M., Krishnan, S., and Silva, C. T.:** Surface Reconstruction based on Lower Dimensional Localized Delaunay Triangulation, *Eurographics 2000*, 20–25, Interlaken, Switzerland.

[23] **Hagen, H., Müller, H. and Nielson, H. M.:** Focus on Scientific Visualization, *Springer Verlag*, 1991, 139–189.

[24] **Heckel, B., Uva, A.E., Hamann, B. and Joy, K.I.:** Surface Reconstruction Using Adaptive Clustering Methods, *Geometric Modelling*, 1999, 199–218

[25] **Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. and Stuetzle, W.:** Surface Reconstruction from Unorganized Points, *Proceedings SIGGRAPH '93*, 1993, 19–26.

[26] **Isselhard, F., Brunnett, G. and Schreiber, Th.:** Polyhedral approximation and first order segmentation of unstructured point set, *In Proc. CGI '98, Hannover, Germany*.

[27] **Jensen, H. W.:** Global illumination using photon maps, *Proceedings Seventh Eurographics Workshop on Rendering*, June 1996.

[28] **Johnson, R. and Wichern, D.:** Applied Multivariate Statistical Analysis (Fourth Edition), *Prentice-Hall, Upper Saddle River, New Jersey*, 1998.

[29] Kós, G.: An algorithm to triangulate surfaces in 3D using unorganized point clouds, *Computing Suppl 14*, May 2001, 219–232.

[30] **Lukács, G., Marshall, A. D. and Martin, R. R.:** Geometric least-squares fitting of spheres, cylinders, cones and tori, *Working paper of the Computer Science Department, University of Wales*, Cardiff, University of Wales, 1997.

[31] **Martin, R.R. and Várady, T.** Estimation of Principal Curvatures from Range Data, *RECCAD Deliverable Documents 2 and 3 Copernicus Project No. 1068, Report GML 1997/5, Computer and Automation Institute, Hungarian Academy of Sciences*, Budapest, 1997.

[32] **Mencl, R. and Müller, H.:** Interpolation and Approximation of Surfaces from Three-Dimensional Scattered Data Points *Research Report No. 662/1997*, December 1997.

[33] **Maneewongvatana, S. and Mount, D.M.:** Analysis of Approximate Nearest Neighbor Searching with Clustered Point Sets, *ALENEX 99*, 1999.

[34] **Pito:** A Registration Aid, *International Conference on Recent Advanced in 3D Digital Imaging and Modelling*, 1997, 93–100.

[35] **Pottmann, H., Chen, H.Y.:** Approximation by Ruled Surfaces, *Journal of Computational and Applied Mathematics 102*, 1999, 143–156.

[36] **Press, W.H., Flannery, B.P., Teukolsky, S.A. and W. T. Vetterling:** Numerical Recipes in C: The Art of Scientific Computing, *Cambridge University Press*, Cambridge, 1988.

[37] **O'Rourke, J.:** Computational Geometry in C, *Press Syndicate of the University of Cambridge*, 1993.

[38] **Ottmann, T. and Widmayer, P.:** Algorithmen und Datenstrukturen, *Wissenschaftsverlag*, 1990.

[39] **Schreiber, Th., Brunnett, G. and Isselhard, F.:** Two approaches for polyhedral reconstruction of 3D objects of arbitrary genus, *International Journal of Vehicle Design, Vol. 21*, 1999, 292–302.

[40] **Vančo, M., Brunnett, G. and Schreiber, Th.:** A hashing strategy for efficient *k*-nearest neighbors computation, *Proceeding CGI 1999*, 1999, 120–128.

[41] **Vančo, M., Brunnett, B. and Schreiber, Th.:** A Direct Approach Towards Automatic Surface Segmentation of Unorganized 3D Points, *Proceedings SCCG 2000*, 2000.

[42] **Vančo, M. and Brunnett, B.**, Consistent orientation of segmented models recovered from digitized data, *Proceedings WSCG 2001*, Pilsen 2001.

[43] **Vančo, M. and Brunnett, B.**, Direct Segmentation for Reverse Engineering, accepted for: *Cyberworlds 2002*, Tokio, Japan, November 2002.

[44] **Várady, T., Martin, R.R. and Cox, J.:** Reverse Engineering of Geometric Models - An Introduction *Computer-Aided Design, Vol. 29*, April 1997, 255–268.

[45] **Várady, T., Kós, G., Benkö, P.:** Reverse Engineering Regular Objects: Simple segmentation and surface fitting procedures, *IJSM (International Journal of Shape Modeling), (Procs. of CAGD: New Trends and Applications, 1997)*, Vol.4, 1998, 127–142.

[46] **Zhang, J. Z. and Chen, L. H.:** Nonmonotone Levenberg-Marquardt Algorithms and Their Convergence Analysis, *Journal Of Optimization Theory And Application*, Vol. 92, Feb. 1997, 393–418.